Mumps Programming Language Interpreter, Compiler, and C++ Class Library User's Guide Including PostgreSQL and MySQL Global Array Database Storage Facility Version 17

Kevin C. O'Kane

kc.okane@gmail.com

http://threadsafebooks.com/
http://www.cs.uni.edu/~okane/

June 17, 2017

Table of Contents

Index of Figures

1 Installation

1.1 Installation Overview

1.2 Interpreter vs Compiler

Please do not use the Mumps **compiler**. It has not been updated and there are possible errors. If you insist upon using it, do not send error reports. It will be brought up to date in a later release. Use the Mumps **interpreter** instead.

1.3 Required System Software

Building mumps requires that your system have certain software installed. For the most part, these are available through the Synaptic Package Manager or *apt-get*. Installation of the *MySQL*, *PostgreSQL*, and *Cygwin* software is only required if you will be using these packages (details below). Normally, you probably won't.

1. Linux, preferably a Debian based version such as Debian, Ubuntu or Mint.

2. The *g++/gcc* compilers and related libraries.

3. The *pcre* (Perl Compatible Regular Expression) development libraries. The pcre libs should be in */usr/lib* and the include files in */usr/include*. Be certain to install the *pcre* **development** libraries.

4. *PostgreSQL* and/or *MySQL (optional)* to store global arrays. Installation *must* include the client **development** libraries.

5. The *bash shell* interpreter located in */bin*.

6. The *GNU readline* and *readline-dev* packages.

7. *Autoconf*

8. The following libraries are needed for the extended precision mathematics. If they are not installed by default, you will need to do so. Be sure to install the **development** versions of the libraries:

   a) The GNU Multiple precision floating point computation library
      http://www.mpfr.org/
      libmpfr-dev

   b) The GNU Multiprecision arithmetic library development tools
      https://gmplib.org/
      libgmp-dev

9. Cygwin DLLs (see below) for use with Windows if you build a windows version of the code. These are not required for Linux versions.

1.4 Basic Software Installation

There are Bash script files (see below) that will install any needed software. You may wish to use these rather than manually installing each software package. The names of these files all begin with the prefix *Configure*. A related set of files to compile and build various versions, begin with the prefix *Build*.

The following are the *apt-get* tool install commands for required software used by Debian GNU/Linux and related distributions (such as Ubuntu and Mint). Other Linux systems use different but similar tools. You need to install these packages for all versions of Mumps[1]:

1 Note: these are automatically installed if you use *ConfigureMysql.script* , *ConfigureNative.scrip*t, *ConfigureNativeClientServer.scrip*t, or *ConfigurePostgresql.script* -- see below for details.

```
apt-get -q -y install autoconf

apt-get install libreadline6 libreadline6-dev

apt-get -q -y install libpcre3
apt-get -q -y install libpcre3-dev

apt-get -q -y install g++
apt-get -q -y install gcc-doc

apt-get -q -y install libgmp-dev
apt-get -q -y install libmpfr-dev
```

1.5 PostgreSQL Software

Install these packages if you will be using PostgreSQL to store the global arrays. These are automatically installed if you use *ConfigurePostgresql.script.*

```
VERSION=9.3
apt-get update
apt-get -y install postgresql
apt-get -y install libpq-dev
apt-get -y install libpq5
apt-get -y install postgresql-doc-$VERSION
```

1.6 MySQL Software

Install these packages if you will be using MySQL to store the global arrays. These are automatically installed if you use *ConfigureMysql.script.*

```
apt-get -q -y install mysql-client
apt-get -q -y install mysql-server
apt-get -q -y install mysql-common
apt-get -q -y install mysql-utilities
apt-get -q -y install mysql-workbench
apt-get -q -y install libmysqlclient-dev
apt-get -q -y install libmysqld-dev
```

1.7 Windows (Cygwin) Software

These DLLs are required for the Cygwin for Windows version (see section 1.11).

```
cygcrypto-1.0.0.dll
cygpcre-1.dll
cygstdc++-6.dll
cygz.dll
cygmysqlclient-18.dll
cygssl-1.0.0.dll
cygwin1.dll
cyggcc_s-1.dll
```

1.8 Building the Software

The distribution consists of source code. The source code must be compiled and linked to create executable versions of the interpreter. There are several options that must be set before compilation. To set these, there is a program named *configure* which can be used to set all the possible options.

However, for the most part, you will use Bash script files that will invoke *configure*, configure the source code, and build the resulting executables according to pre-set templates. Each of these begins with prefix *Build*. They are discussed below.

1.8.1 Quick Start

If you want to build the most basic version of the Mumps interpreter, run:

```
ConfigureNative.script
```

followed by:

```
BuildMumpsWithGlobalsInSingleUserNative.script
```

The first script file installs any necessary software and the second compiles and builds the most basic version of the interpreter. If you have already installed the necessary software, the first step is not needed. You must be **root** to run these scripts.

The resulting interpreter is named *mumps* and is located in /usr/bin/mumps.

1.9 Databases

The *configure* program tailors the code to set a number of options most of which you will probably not change (see section 1.14 on page 24 for a complete list of options). Included in the distribution are a set of *bash* scripts that configure and build Mumps.

This Mumps distribution has four options with regard to storing the global arrays:

1. Store the global arrays in the native B-tree database.

    a) single user version
    b) multi-user shared database version
    c) multi-user client-server version

2. Store the global arrays in a local or remote PostgreSQL data base.

3. Store the global arrays in a local or remote MySQL data base.

Option 1, referred to as the native database, is quite fast with a minimum of overhead and it can efficiently manage very large databases but it lacks a number of features normally found on modern database systems. It is sensitive to system and programming errors. It does a minimum of checkpointing and maintains a large part of the global array tree in volatile memory. If the host system crashes or the program using the global arrays terminates unexpectedly, the contents of the entire global array database are likely to be lost.

However, in applications where speed is important and, in the event of a crash, the program can be re-run without loss of data, the native database is a good choice.

The native database has two configurations. The first of these is a *single user* global array facility where the global arrays are stored in one directory, usually the one in which the Mumps program is itself running. In this mode, only one *read-write*[2] Mumps program may access the global arrays in a given directory at a time although other Mumps programs may run concurrently in other directories operating on other global array data sets. This is the fastest but most restrictive option.

A second native database configures involves running a local global array server. The server accepts global array access requests by means of internal operating system pipes from one or more Mumps client programs operating concurrently. This option is slightly slower (about 30%) due to additional system overhead but it permits multiple concurrent program execution. Only clients running on the same machine as the server, however, may access the database as there is no networking option. The server's global array files are slightly better protected as the server is not affected by client program crashes and the server does periodic flushing of it's buffers and journaling is possible. However, synchronization of array access is more tricky and requires use of the awkward and error prone Mumps **lock** command. Also, the global array database may be lost due to host system crash or failure of the Mumps server.

If data integrity, remote and multi-user access are important, options 2 and 3 are better. These use PostgreSQL and MySQL, respectively, to store the global arrays. However, while the global array access is slower there are several other advantages.

2 The native database Mumps comes in two versions: a *read-write* version which may both read and write global arrays and an *read-only* version where each Mumps program may only read the global arrays. Multiple *read-only* instances may operate concurrently on the same global array data sets.

While options 2 and 3 are slower than option 1, due to relational data base system overhead, using a relational database has *significant advantages* with regard to reliability and flexibility. These include:

1. All database transactions are ACID (*Atomicity, Consistency, Isolation, Durability*) compliant.
2. SQL commands such as Begin Transaction, Commit and Rollback are available.
3. The Mumps global arrays can be queried with SQL commands from non-Mumps environments.
4. SQL views of the Mumps database may be constructed.
5. The Mumps global array database can be remote and distributed.
6. Mumps programs can execute SQL commands on the server on any accessible database table.
7. Multiple concurrent Mumps programs may run at the same time.

The distribution contains several scripts that will build various versions of the system. These are detailed next. You must be *root* to run these.

The scripts assume a Debian (*apt-get*) based Linux installation. If you are using a version of Linux not based on Debian, you will need to install and configure the required system software manually according to the procedures on your system.

Some of the scripts provided with the distribution may install system software as needed. Consequently, when using these scripts, your machine needs to have a reliable Internet connection. Also, due to Internet load factors, it is possible that software installations may take a long time or, in some cases, fail in the unlikely event that the servers from which the software to be downloaded are unavailable.

The Mumps interpreters and libraries built as a result of the scripts will be stored in */usr/bin, /usr/lib, /usr/include* and, in the case of the native file system server code, */etc/mumps*.

For the most part, except for the Cygwin build, there are two scripts for each option. One of these installs all necessary system software and then builds the Mumps interpreter. The second only configures and builds Mumps, it does not install system software.

In the case of the scripts used with the relational databases, you will need to know the root password for the relational system if it is already installed or, if the script installs it, you will need to give the relational system a root password. In the case of the relational databases, the scripts will create a user named *mumps* and a database named *mumps*. The *mumps* user is granted superuser rights.

1.9.1 Native B-tree Database

1.9.1.1 Native B-tree Client-Server Database

```
ConfigureNative.script
BuildMumpsWithNativeClientServer.script
```

These build the client-server native database global array file system.

The script ultimately create two binary executable programs: *mumpsd,* which is the database server, and *mumps,* which is the interpreter client. See below for details.

The first script, *ConfigureNative.script*, will install any missing system software and then you invoke the second script (*BuildMumpsWithNativeClientServer.script*) which actually compiles and installs the Mumps programs.

If required system software is already installed, you only need to use the second script, *BuildMumpsWithNativeClientServer.script*

If required system software is not installed, you must first install the software either manually or by use the first script.

The executable Mumps client interpreter is place in */usr/bin* while the Mumps global array server is placed in */etc/mumps*.

The Mumps global array server permits multiple, local Mumps client programs to access the global arrays concurrently. There is approximately a 30% performance penalty on global array accesses done through the server as opposed to accesses using the stand-alone, single user file system.

The build scripts described above will create the global array libraries so that global array accesses will be made through the mumps server (*mumpsd*).

To start the server, change directory to */etc/mumps and* start the server, as root, with the command

```
./mumpsd > log &
```

To halt the server, send it a *SIGINT* (^C) such as[3]:

```
kill -2 1234
```

where 1234 is the *mumpsd* process id. If you do not start the Mumps server, global array access will be unavailable.

If you do a proper Mumps demon shutdown, the database will be intact. However, failure to properly close the server could lead to catastrophic data loss.

The server communicates with the clients by means of sockets.

The Mumps global array data base (*key.dat* and *data.dat*) will be in */etc/mumps*.

On some multi-core systems, a slight performance improvement may be gained by attaching the Mumps demon to one CPU. To do so, use the following command as root:

```
schedtool -a 0x1 PID
```

where PID is the process id of the demon and 0x1 means the first (cpu0) processor core (0x2 means cpu1 *etc.*).

1.9.1.2 Native B-tree Database witout Server

```
ConfigureNative.script
BuildMumpsWithGlobalsInSharedNative.script
BuildMumpsWithGlobalsSingleUserNative.script
```

These scripts build two stand-alone versions of the mumps interpreter both of which use the native B-tree global array file system, the fastest global array database option.

When using a native btree global array database, the database is stored in two files: *key.dat* (the B-tree) and *data.dat* (the stored data). Normally these reside in the same system directory as the executing Mumps program.

A given system may have multiple global array databases in multiple directories. Each such database is completely separate and independent from each other database.

1.9.1.2.1 Shared Database

BuildMumpsWithGlobalsInSharedNative.script creates a version that does not require a server. Instances of the Mumps interpreter share the database.

In shared mode, there may be multiple instances of Mumps interpreters operating concurrently on the same database files (*key.dat* and *data.dat*). They cooperatively share the database. Each instance is given a *slice* of operations on the database before it relinquishes control. The size of this slice can be set by *configure* (see page 24).

3 Note: during normal Linux shutdown, this signal is automatically sent to each process so *mumpsd* will be properly shutdown.

1.9.1.2.2 Single User Mode

The script *BuildMumpsWithGlobalsSingleUserNative.script* creates a single user version of Mumps. The database in read-write mode is not shared. Programs must acquire access to the database serially. This is the fasted version of the database.

Because many applications tend to write or update the database infrequently but read it frequently, one version of the Mumps interpreter built by this script is *read-only* with respect to the database while the other is r*ead-write.* In the read-only version (*mumpsRO*), the same database may be accessed by any number of Mumps programs concurrently.

1.9.2 Relational Database Server Resident Global Arrays

1.9.2.1 Overview

The Mumps global arrays my be stored in a relational database system. The two currently supported are MySQL (Oracle Corporation) and PostgreSQL (PostgreSQL Global Development Group). With simple code changes, other servers could also be accommodated.

There are advantages and disadvantages to storing globals arrays in a relational database. The hierarchical nature of the Mumps database is ordinarily not well suited to the tabular structure of a relational database and access is slower.

On the other hand, relational databases provide flexible multi-user, robust, fully ACID (*Atomicity, Consistency, Isolation, Durability)* compliant data storage along with a complete suite of transaction processing functions not otherwise available in the Mumps language definition.

A further advantage is that global array data may be interrogated and manipulated by ordinary, standard SQL commands.

By default, the Mumps interpreter maps global array references to a multi-column relational database table with the same name as the global array. The columns of the table are named *a1*, *a2*, … *a10* and so forth. The values in the columns are the indices from a global array reference.

The final column contains the value stored at the reference, if any. For example, the code:

```
set ^birds(1,2,3,4,5)=”ducks”
```

would map to a table named *birds* in the relational database as follows[4]:

birds

```
+------+------+------+------+------+------+
| a1   | a2   | a3   | a4   | a5   | a6   |
+------+------+------+------+------+------+
| 1    | 2    | 3    | 4    | 5    |ducks |
+------+------+------+------+------+------+
```

The total number of columns for a global array is set either to a default number (a *configure* option) or by the *sql/f* Mumps command. The *sql/f* command for the above would look like:

```
sql/f birds 6
```

where the first operand is the table name to be created and the second is the number of columns (including the final column for data values).

If you do not predefine a table, a default number of columns will be used (currently 11).

If your program instantiates array elements like the following:

```
set ^birds(1)=”all”
```

4 By default, the columns *varchar* (note: the character length is a settable option but the index columns are normally *varchar(64)* while the data column, the last column, is normally *varchar(512)*). The character size of columns can be set to other values by *configure*. Smaller values may improve performance.

```
set ^birds(1,2)=”flying”
set ^birds(1,2,3)=”water”
set ^birds(1,2,3,4)=”large”
set ^birds(1,2,3,4,5)=”ducks”
set ^birds(1,3)=”flightless”
set ^birds(1,3,3)=”water”
set ^birds(1,3,3,4)=”large”
set ^birds(1,3,3,4,5)=”penguins”
```

The relational table will look like[5]:

birds

a1	a2	a3	a4	a5	a6
1	(null)	(null)	(null)	(null)	all
1	2	(null)	(null)	(null)	flying
1	2	3	(null)	(null)	water
1	2	3	4	(null)	large
1	2	3	4	5	ducks
1	3	(null)	(null)	(null)	flightless
1	3	3	(null)	(null)	water
1	3	3	4	(null)	large
1	3	3	4	5	penguins

Mumps access requests produce the expected results:

```
write ^birds(1)             => all
write ^birds(1,2)           => flying
write ^birds(1,2,3)         => water
write ^birds(1,2,3,4)       => large
write ^birds(1,2,3,4,5)     => ducks

write $order(^birds(1,2))    => 3
write $order(^birds(1,2,"")) => 3
```

The row-wise duplication in the above is also present in many other Mumps systems and the *nulls* have little effect on overall performance. Ideally, however, a more dense table is easier to deal with from a SQL frame of reference.

An advantage, as mentioned above, is that data stored in such a table may be queried by an ordinary SQL command such as:

```
select a6 from birds where a1='1' and a2='2' and a3='' and a4='' and a5='';
```

which yields *flying*. Note the specific use of null strings.

Similarly, SQL *views* may be established on the *birds* table to facilitate access.

1.9.2.2 Basic Database Configuration

By default, in order for Mumps to store and retrieve global arrays from a relational server the following requirements must be met:

1. There must be a database user named *mumps* authorized to create, drop, read and write tables;

5 Table row order may differ but this is not important.

2. There must be a database named *mumps* with privileges sufficient to create, destroy, read and write tables.

The user name *mumps* and the database name *mumps* may be changed in the *configure* procedure. By default, user *mumps* has the default password *abc123* which may also be changed with *configure.*

Note: if you want to experiment with this before committing it to your main host, you might try building a virtual Linux machine with Linux Mint and Oracle's Virtual Box (both are free).

1.9.2.3 Relational Database Configure Options Common to All Servers

The following *configure* options are common to all relational database clients. You probably do not want to change these.

1.9.2.3.1 --with-datasize=*numeric-value*

The maximum length of a string stored at a global array node in the last column. Performance is improved if this value is as small as possible. If an element stored at a global array node exceeds this length, it will be truncated. Default: 512.

1.9.2.3.2 --with-dbname=*name*

The name of the *database* in the relational database server where Mumps will store the global arrays. Default: *mumps.*

1.9.2.3.3 --with-indexsize=*numeric-value*

Specify the size of the *varchar* declaration of columns *a1, a2, ...* This is the maximum string length of any individual global array index element. For example, if your global array reference is: *^a(1,2,3),* you have three index columns, (*a1, a2,* and *a3*) and a data columns (*a4*). This option sets the maximum string length of columns *a1, a2,* and *a3.* Performance is improved if this value is as small as possible. If an individual element of a global array index exceeds this length, it will be truncated. Default: 64.

1.9.2.3.4 --with-tabsize=*number*

Maximum number of index elements in a global array reference not counting the name of the global array itself. Default: 10. This is the maximum depth of any global array tree. The maximum permitted value is 31.

1.9.2.4 Initialization of a Mumps Relational Database

The ***first time*** you use a table in the relational database, you ***must*** initialize it with the Mumps command:

```
sql/f global_name columns
```

where *global_name* is the name of the global array. When you do this, you may see a warning message that the *global_name* table does not exist. This can be ignored. A second running of the initialization command will not show the message[6].

If you do not initialize a gloabl array before using it, a default ten column arrray will be constructed.

1.9.3 Relational Database Server Connection for Mumps Global Arrays

In order to store the global arrays in a database server, you need to create a user named *mumps* and a database named *mumps* (these defaults can be changed by means of *configure*).

You also need to inform the Mumps client code of the password for the *mumps* user. This can be set by means of *configure* or in *btree.cpp.in* (in the section corresponding to the server (MySQL or PostgreSQL) you are using - see below). The default password for user *mumps* is *abc123*.

For PostgreSQL, the connection information, including *user*, *database* and *password*, is found in function *AllocSV()* in file *sysfunc.cpp.in*:

6 It is due to an *SQL DROP* statement on the table relation before table is built.

```
strcpy(p1->Connection,"host=@remotehost@ dbname=@dbname@ user=@user@
      password=@passwd@  ")
```

For MySql, this information is found in file *btree.cpp.in* the code*:*

```
      char host[128]="@host@";
      char user[128]="@user@";
      char passwd[128]="@passwd@";
      char dbname[128]="@dbname@";
      unsigned int port=@port@;
      char socket[128]="@socket@";
```

Items enclosed in @-signs are replaced by *configure*. The values refer to connection options for the respective database servers. The defaults are database name: *mumps;* user: *mumps;* password: *abc123;* and host: *localhost*.

These options can be set by *configure*.

1.9.4 MySQL Database Option

```
ConfigureMysql.script
BuildMumpsWithGlobalsInMySQL.script
```

The script *ConfigureMysql.script* (1) installs any required system software (including MySQL), configures MySQL, and (2) invokes the second script to compile and build the MySQL client *mumps* interpreter.

The MySQL required software is listed in section 1.6 on page 8.

BuildMumpsWithGlobalsInMysql.script compiles and builds a MySQL client *mumps* interpreter. It assumes that required system software has already been installed.

The MySQL client Mumps interpreter requires a properly configured and running MySQL server for global array access.

The script *ConfigureMysql.script* is mainly useful as a quick start for an installation which does not already have a MySql server running on it. If the script detects the directory */usr/lib/mysql* it assumes that MySql is present and does not attempt to install MySql. It proceeds, instead, to determine if any system software is missing and install as necessary. It then invokes the second script.

If you have an existing MySql server, you may want to manually set up the necessary *mumps* user and password along with any missing system software.

Depending on whether your host machine will be client, server, or both, you will need to install the appropriate MySQL software. This will include the appropriate development libraries. If you receive an error message during compilation of link edit, it is probably due to missing libraries.

1.9.4.1 MySQL Manual Installation

The script file *ConfigureMysql.script* can be used to install and configure Mumps and MySQL on a system that does not already have MySQL or Mumps installed. It identifies and installs missing system software, configures Mumps, and initializes the MySQL database.

The script assumes you are working on a version of Linux (such as Mint, Ubuntu, Debian, etc.) that uses the *apt-get* to install and upgrade software. If you are not, you will need to manually install the packages itemized in the script.

If you already have MySQL installed, this script will install any parts that are missing (such as the development libraries) and upgrade others. The script will also install or upgrade other system software needed by Mumps.

If you do not want some or all of the packages upgraded, do not use this script. Manually install the missing software.

If MySQL was not previously installed, you will be prompted to give a password to be used for the MySQL root user. You must do so since you will need this password later in the procedure. The installation will otherwise use standard MySQL defaults.

After the software is installed, the script to configure Mumps is called (*BuildMumpsWithGlobalsInMySQL.script*). If, after the initial MySQL installation, you need to change a configuration setting in Mumps, you may invoke this script directly. Its options are given in section 1.9.4.2 on page 16 below.

1.9.4.2 MySQL Installation Options

The following are the *configure* options for MySQL and their default values.

1. --with-mysql-user=user

   The MySQL userid of the client Mumps program to be used when establishing a connection. May be set in the **sql/d** connection string. Default: *mumps*

2. --with-mysql-host=nbr

   The IP number of the MySQL server. May be set in the **sql/d** connection string. Default: *localhost*

3. --with-mysqldb

   Enables MySQL database storage of globals. MySQL is not enabled unless this option is specified. May not be set by **sql/d** connection string. Incompatible with the corresponding PostgreSQL enabling option.

4. --with-mysql-passwd=val

   Specify, if needed, the MySQL user passwd. May be set in the **sql/d** connection string. Default: the *empty string* (no password).

5. --with-mysql-port=nbr

   Port number to access the MySQL server. Default: 0.

6. --with-mysql-socket=nbr

   Socket through which to access the MySQL server. Default: NULL.

1.9.4.3 Mumps Build for MySQL Resident Global Arrays

The script *BuildMumpsWithGlobalsInMySQL.script* contains the code to build and configure Mumps to use a MySQL server. This script also sets the MySQL option for the compiler and the toolkit.

The script file *ConfigureMysql.script* can be used to install and configure MySQL software. It then invokes *BuildMumpsWithGlobalsInMySQL.script.* After you have installed MySQL, use *BuildMumpsWithGlobalsInMySQL.script* to make configuration changes.

1.9.4.4 Configuring a Remote MySQL Server

See MySQL documentation.

1.9.4.5 Using MySQL Resident Global Arrays from Mumps

Same as for PostgreSQL with the exception of differences in the connection string used by **sql/d**.

1.9.4.6 Command Line Interpreter *mysql*

The MySQL command line interpreter is named *mysql*. You may invoke it for user *mumps* and password *abc123* with:

```
mysql -u mumps -pabc123
```

To use the *mumps* database, type:

```
use mumps;
```

Other options are:

1. *show tables;* Displays the tables in the database.
2. *show columns from abc;* Displays the columns from table *abc.*
3. *set global innodb_flush_log_at_trx_commit=0;* Improves transaction speed at the expense of reliability (see MySQL documentation).

Note: in MySQL you may not use some table names. One of these is *index*.

1.9.4.7 Kill Command in MySQL

In MySql, a table must exist before you can delete it's contents. Thus, if you execute a command such as:

```
Kill ^A
```

Where global array variable *^A* does not exist (that is, there is no MySQL table for the variable), MySQL will generate an error message and your program will halt.

1.9.5 PostgreSQL Database Option

```
ConfigurePostgresql.script
BuildMumpsWithGlobalsInPostgreSQL.script
```

The script *ConfigirePostgresql.script* installs required system software (including PostgreSQL), configures PostgreSQL, and then invokes the second script to compile and build a PostgreSQL client *mumps* interpreter.

The second script, *BuildMumpsWithGlobalsInPostgreSQL.script*, builds a Mumps PostgreSQL client. The PostgreSQL client mumps interpreter requires a properly configured and running PostgreSQL server if access is made to a global array. After you have installed PostgreSQL, use *BuildMumpsWithGlobalsInPostgreSQL.script* to make any configuration changes.

The required PostgreSQL software is listed in section 1.5 on page 8.

Note: In PostgreSQL, tables names are stored in lower case. Thus, a table or Mumps global array named AAA is the same as aaa.

In general, overall performance appears to be better if you use PostgreSQL rather than MySQL.

1.9.5.1 PostgreSQL Options

Mumps permits storage of global arrays in PostgreSQL database tables. Using PostgreSQL gives the Mumps user a fully ACID (*Atomicity, Consistency, Isolation, Durability)* compliant database but database access will be slower overall.

When you create/store global arrays, they will be stored, by default, in a database known as *mumps* on the PostgreSQL server. The tables created in this database will have the same names as the corresponding Mumps global arrays and may also be accessed from non-Mumps clients by means of SQL *SELECT* and related statements.

When using a PostgreSQL server, it is possible to construct views of database tables so that they can be directly accessed by Mumps as global arrays. An example of this is given below.

When storing global arrays on a PostgreSQL server it may be desirable, when beginning a series of related global *create/store/update* transactions, to precede the Mumps code with:

```
SQL BEGIN TRANSACTION;
```

ultimately to be followed by:

```
SQL COMMIT;
```

This permits the Mumps global array create/store/updates to run faster. It also insures that the all the transactions will run without interference from other users. This eliminates the need for the Mumps **lock** command.

However, should there be a failure before the final *COMMIT*, the uncommitted data may be lost.

1.9.5.2 Command Line Interpreter *psql*

Assuming the default database configuration (see *Quick PostgreSQL Installation* in section 1.9.5.3.1), you may access the PostgreSQL command line interpreter with:

```
psql mumps
```

From this program, you may access the Mumps tables. Also note that there is a modified version of *psql* that permits execution of Mumps commands from the CLI.

1.9.5.3 Installing PostgreSQL

Note: as of this writing, the PostgreSQL release is 9.3 and this number is used in the following documentation. Check which version you have and adjust the following accordingly. Subsequent versions of PostgreSQL may have different interfaces and may use different libraries which may invalidate some or all of the following.

1.9.5.3.1 Quick PostgreSQL Installation

If your system is Ubuntu-based and you do not presently have PostgreSQL installed or you have a more or less standard PostgreSQL installation, there is a script file in the distribution that should be able to build and install both Mumps and PostgreSQL. The script is named:

```
ConfigurePostgresql.script
```

You will want to edit this script to tailor it to your needs. Instructions are contained in the comments. This script *must* be run as *root*. It was developed using Linux Mint 17.2 and PostgreSQL version 9.3 and should work with related distributions with a change of version number as needed.

1.9.5.4 PostgreSQL Configuration

You may want to modify some PostgreSQL configuration options. These configuration options are usually found in:

```
/etc/postgresql/9.3/main/postgresql.conf
```

Note: replace *9.3* but the current version of your installation.

For example, in order to suppress extraneous notices (as opposed to warning and error messages) from appearing in the output of a PostgreSQL client (e.g., Mumps), you may want to set the following configuration parameter:

```
client_min_messages = warning
```

1.9.5.4.1 PostgreSQL Specific Mumps Install Options

The following detailed setup instructions apply to Ubuntu and Ubuntu-like distros such as Mint. In Red Hat based distros, some PostgreSQL files may be located in different directories and this may affect the installation procedures. The instructions here are based on Linux Mint Mate 17.

If you have not installed PostgreSQL or only have a basic PostgreSQL installation, see section 1.9.5.3.1 above which does most of the following automatically.

The primary purpose of the following is to create a database named *mumps*, a PostgreSQL user named *mumps* with an initial password of *abc123*. The Mumps PostgreSQL, by default, logs into the server as *mumps* and creates it's relational tables in the *mumps* database.

1. Using your package manager (e.g., Synaptic) install the latest version of PostgreSQL including the development libraries. See section 1.5 on page 8.

2. Configure PostgreSQL options are:

```
configure prefix=/usr \
--with-pgdb=/usr/include/postgresql \
--with-dbname=mumps \
--with-pgsql-host=127.0.0.1  \
--with-pgsql-user=mumps \
--with-pgsql-passwd=abc123
make
make install
```

Note: different versions of PostgreSQL have had a habit of playing *hide the files* which may cause problems. Check for updates if you experience problems. The line:

```
--with-pgdb=/usr/include/postgresql
```

Tells *configure* where the include files are located. This is currently the default location.

The other options indicate:

1. *prefix*: where to place the Mumps executables.
2. *--with-dbname:* the name of the database in which to store the global arrays.
3. *--with-pgsql-host:* the IP number of the machine hosting the server (127.0.0.1 is *localhost*)
4. *--with-pgsql-user:* the user name that Mumps will login to the server as.
5. *--with-pgsql-passwd:* the password to be used by Mumps when logging in.

The sever ***must*** be running and properly configured in order for the Mumps global array facility to function[7].

Once you have built a Mumps database in PostgreSQL, you may query it with general purpose SQL commands (such as *SELECT,* discussed elsewhere).

1.9.5.4.2 Configuring the *listen_address*

By default, Mumps logs into the PostgreSQL server on the *current* machine. If you want to run Mumps programs on a different machine than the one running Mumps, you need to enable, on the *server* machine, connections to its PostgreSQL server.

The following are some brief instructions on how to permit a remote server to process Mumps requests. You should consult the PostgreSQL manuals for details which would be more appropriate to your application:

7 *Note:* it appears that some Synaptic package manager installs *may* incorrectly address the location of the Postgresql socket. If, upon starting mumps, you get a message that the connection could not be opened and to check the socket, you will need to correct an entry in the file:

```
/etc/postgresql/9.3/main/postgresql.conf
```

In this file, change the value for the entry *unix_socket_directory* to point to the directory in the error message (probably: */var/run/postgresql*). This is a PostgreSQL issue, not a Mumps issue. You will need to restart the database after this or any other configuration changes:

```
/etc/init.d/postgresql restart
```

To accept connections, you should set, on the host machine, as root, the ***listen_address*** option in the file:

```
/etc/postgresql/9.3/main/postgresql.conf
```

to contain the IP numbers of the systems from which you are willing to accept connections. Note: the intermediate directory *9.3* in the above refers to the current PostgreSQL release number. This will change with time.

For example:

```
listen_addresses = 'localhost,*'
```

The above, note the quotes, permits connections from all remote addresses. After altering this setting a restart is required:

```
/etc/init.d/postgresql restart
```

The connecting clients' IP numbers should identified be in the file ***pg_hba.conf*** found in the same directory. To enable a network connection, you should insert a line into this file. If you are using IPV4 addresses, it should look something like:

```
host    all         all         10.42.0.0/16      trust
```

which means that the the high order 16 bits of the IPV4 of the incoming request IP number must match *10.42* but the remaining 16 bits can be any value. The server will accept connections from any machine with the *10.42* prefix. The *trust* option means that a password will not be required from the connecting client. If you want the user to supply a password, use:

```
host    all         all         10.42.0.0/16      md5
```

There are other security options. Consult the PostgreSQL documentation or your system administrator.

You can test the connection to the server from a remote machine with the command:

```
psql -h 10.42.0.26 -d mumps
```

where the '-h' option specifies the remote host to connect to and the *-d mumps* specifies the name of the database. This command assumes that the *trust* option was used and that the login_userid of the user on the remote (client) system is the same as an authorized user on the remote (server) system. The server machine is at address *10.42.0.26.* To exit from *psql*, type *\q*. (backslash-q).

If you attempt to use Mumps with PostgreSQL as the database from an account not recognized by PostgreSQL, you will receive the error messages of the form:

```
*** Connection to database server failed in or near line 0
Error msg: FATAL:  role "root" does not exist
Connection string=dbname=mumps
```

The PostgreSQL server can be started with SSL enabled by setting the parameter *ssl* to **on** in *postgresql.conf.*

1.9.5.4.3 Performance Tuning

By default, PostgreSQL is set for stringent data protection. This results in considerable disk activity to insure that data is never lost. However, many of these procedures slow the operation of the database during update to a considerable extent. They can, in many cases, be dispensed with with only minimal effect on database integrity.

The main configuration file is *postgresql.conf*. For a server started by the operating system, this file will, by default, be in */etc/postgresql/9.3/main.*

Alter the settings as follows:

```
wal_level = minimal

fsync = off

synchronous_commit = off

full_page_writes = off

archive_mode = off
```

The result may be a considerable improvement in speed.

In cases where speed is important, a series of inserts into the database that are done as one transaction is faster than individual transactions (the default). For example:

```
1   #!/usr/bin/mumps
2
3    sql/f a 4
4
5    set k=0
6
7    sql SET LOCAL synchronous_commit TO OFF;
8    sql begin transaction;
9
10   for i=1:1:100 do
11   . s k=k+1
12   . s ^a(i)=k
13   . for j=1:1:100 do
14   .. set k=k+1
15   .. set ^a(i,j)=k
16   .. for m=1:1:10 do
17   ... s k=k+1
18   ... s ^a(i,j,m)=k
19   sql commit
20   halt
```

One line 3 the table *a* is cleared and initialized to four columns. On line 7 the SQL command disables the PostgreSQL server's from waiting for the transaction's records to be flushed to permanent storage before returning a success indication to the client. This causes the inserts to proceed very much faster but at some risk of data loss (but not data corruption) should the system fail during updates.

Line 8 initiates a transaction and line 19 commits the transaction that finalizes the values stored in the intervening lines.

1.9.5.4.4 PostgreSQL Server Connections

Each time you start Mumps you ***must*** establish a connection with a PostgreSQL server. The command to do this is *sql/d* but this is normally done automatically the first time a Mumps program references a global array. The default connection settings are normally used.

The *sql/d* command is only used to connect with a server other than the default or to reconnect with the default server if the original connection was terminated.

Normally, a default connection is made automatically when anytime you attempt to use a global array. The defaults are established by *configure.*

The *sql/d* command should only be used when you are making a non-default connection.

The prototype for the connection command sent by a Mumps program (found in the code module *sysfunc.cpp.in*) to the server is:

The *sql/d* command is used when you want to make a connection using connection parameters other than those established as defaults. Normally, you will not use this command.

Sql/d may be entered in interactive mode or as part of a Mumps script file. If you want parameters other than the defaults, you must place them on the *sql/d* line and execute the line *before* any attempt to access the global arrays. Once you attempt to access a global array, the default connection is automatically established.

The options available in *sql/d* and their keywords are the same is given in the PostgreSQL documentation for the *PQconnectdb()* function.

The *sql/d* command causes any current database connection to be closed. The remainder of the line consists of the new connection arguments, some of which may be of the &~*exp*~ format[8]. If successful, **$t** will be true. This connection information will be used until changed or the Mumps client terminates. Examples:

```
sql/d dbname=mumps

sql/d host=abc.def.xyz.edu dbname=mumps

sql/d hostaddr=123.321.432.321 dbname=mumps

sql/d user=joe password=abc123 dbname=mumps host=abc.def.xyz.edu
```

1.9.5.4.5 PostgreSQL Transaction Limit

The number of pending transactions between SQL Begin and SQL Commit commands is limited by PostgreSQL. If this number (implementation defined) is exceeded, transactions will be lost. Check the PostgreSQL log in */var/log/postgresql*.

1.10 Use with Apache

When running through the CGI-BIN interface with the Apache web server, be sure your files and directories are not owned by *root*. Make them owned by Apache (user *www-data* in Linux Mint 13) and in the Apache group (also *www-data*). Apache's default cgi-bin directory is /usr/lib/cgi-bin. You will need to be *root* to add/modify files in this directory. Be sure to make Apache (*www-data*) an authorized PostgreSQL user with the *createuser* command (see 1.9.5.4.1 above).

1.11 Use with Windows

If you install all the relevant Cygwin software, you may build a native database version of Mumps in Windows[9] using Cygwin for the native and relational database versions. An executable (*mumps.exe*) so built, and may be copied to another directory and executed from a normal Windows command prompt ***if*** you copy several Cygwin DLLs to your system (\Windows) directory. These are[10]:

```
cygcrypto-1.0.0.dll
cygpcre-1.dll
cygstdc++-6.dll
cygz.dll
cygmysqlclient-18.dll
cygssl-1.0.0.dll
cygwin1.dll
cyggcc_s-1.dll
```

The first group is needed for all versions while the second group is also needed for MySSQL versions. These can Be found in the Cygwin */bin* directory.

The *cygmysqlclient-18.dll* is only needed for the MySQL build. (Note: version numbers may bedifferent when you read this).

If you build a MySQL version in Cygwin, even though it is running under Cygqin, Mumps will access the MySQL server running on Windows (or at a remote machine if you specify a remote IP

8 See section 5.17 on page 46.

9 Only limited testing is done on Windows. If you want to run Mumps on Windows, use a virtual machine package such as Oracle's Virtual Box and install Linux Mint Mate.

10 Note: version numbers may change with time.

number). You may run Mumps natively in a Windows command prompt box without Cygwin if you install the DLLs noted above in your Windows folder.

1.11.1 Windows/Cygwin Install

The installation scripts *Build...* will work with *Cygwin* if the appropriate software is installed. However, the configuration scripts *ConfigureNativeClientServer.script*, *ConfigureNative.script*, *ConfigurePostgresql.script* and *ConfigureMysql.script* will not work.

1.12 Math Options

Arithmetic in this Mumps distribution can be performed either by hardware or by a library of extended precision software.

In extended precision mode, the precision of both floating point and integer numbers can be significantly larger than is the case with standard hardware arithmetic with minimal performance penalty.

The several Build scripts look for files *gmp.h* and *mpfr.h*. If these are found, they cause the build to use the extended math packages. If not, the builds will use hardware arithmetic.

You may override this and force hardware arithmetic by modifying the scripts to add the *--with-hardware-math* option.

1.13 Numeric Configuration Options

Both extended precision and basic hardware precision are available as noted above.

1.13.1 Hardware Math

In hardware math mode, integer and floating point numbers are processed by the machine's arithmetic processing hardware. Floating point numbers are treated as either *long double* or *double* values and integers are treated as either signed 64-bit *long long* or signed 32-bit *long* integer values.

To enable hardware math, you must specify the following as a *configure* option:

--with-hardware-math

Integer arithmetic may be performed in *long* (32 bit) or *long long* (64 bit) mode. The default is *long long*. The *long* mode may be turned on with the *configure* option:

--with-int-32

If the above is not specified, *long long* is used.

Floating point arithmetic may be performed in either *long double* or *double* mode. The *long double* mode may be enabled with the *configure* option:

--with-long-double

If the above is not specified, floating point arithmetic will be performed in *double* mode.

All numeric values are stored internally as strings. They are converted to binary numeric integer or floating point format just prior to an arithmetic operation and then converted back to strings.

By default, the string format of a floating point number will have with 8 digits of precision. This can be altered by *configure* using the *--float_digits* option (default is 8). For example, if you want 16 digits of precision, add

--float_digits=16

to the *configure* parameters. The number of digits specified should be consistent with the hardware data type (*double* or *long double*).

On x86 architectures, *long double* is probably implemented as an 80 bit number with a sign bit, an 15 bit exponent and 63 bit fractional part with a range of approximately 3.65×10^{-4951} to 1.18×10^{4932} while *double* is implemented as a 64 bit number.

1.13.2 Extended Precision Math

Extended precision is available through use of the GNU multiple precision arithmetic library[11] and the GNU MPFR library[12]. For integers, this means effectively unlimited precision. For floating point numbers, the exponent is 64 bits and the fraction is user specified (default of 72 bits in Mumps - this option may be set by *configure*).

Hardware arithmetic will be selected during compilation of the interpreter if (1) *configure* does not find the extended precision libraries or the user specifies the configuration option:

--with-hardware-math.

If extended precision is used, the number of bits in the fraction of a floating point number can be set with:

--with-float-bits=value

where *value* is the number of bits. The default value is 72.

For extended precision floating point numbers, the number of digits of precision to print is controlled by:

--with-float-digits=value

where *value* is the number of digits. The default is 8.

When printing a number, the number of digits being printed should be consistent with the number of bits in the fraction. If the number of digits to print is too large, random low-order digits may appear in numbers.

1.14 All Configure Options

The basic install sequence, as is the case with many Linux based packages is to run something similar to the following as *root*:

```
./configure prefix=/usr
./make
./make install
```

The configure step, however, as is typical, contains many options. Specifying these causes modification to the source code and changes the final product.

The distribution, as noted above, contains several *bash* script files with pre-configured *configure* commands. For the most part, you probably don't want to write your own *configure* options except in limited cases. You may, however, want to edit the files provided to set details such as passwords and so on. This is discussed below.

The full set of options to *configure* are:

1. configure prefix=/usr

   The directory where the runtime modules will be stored. If this is not specified, the default location is in a directory named *mumps_compiler* in the user's home directory. Normally, if you want Mumps available to all users, you will specify the option as shown and run *make* and *make install* as *root*. If you specify */usr* as shown, the Mumps routines will be placed in */usr/bin/mumps*.

2. PostgreSQL options

   a) --with-pgsql-user=userid — userid for PostgreSQL server account [mumps]
   b) --with-pgsql-passwd=password — password to access database [“abc123”]

11 http://www.mpfr.org/
12 http://gmplib.org/manual/index.html

c) --with-pgsql-host=IP select host IP number [127.0.0.1]
d) --with-pgdb= location of libraries [/usr/include/postgresql]

3. MySQL options

a) --with-mysql-user=userid userid for MySQL server account [mumps]
b) --with-mysql-host=IP IP number of remote host [127.0.0.1]
c) --with-mysqldb Enable MySQL data base for globals
d) --with-mysql-passwd=val Select mysql user passwd ["abc123"]
e) --with-mysql-port=nbr Select mysql port [""]
f) --with-mysql-socket=nbr Select mysql socket [""]

4. General Relational Database Options

a) --with-dbname=name SQL data base name [mumps]
b) --with-indexsize=number SQL DB index max [64]
c) --with-tabsize=nbr number of columns in SQL table *mumps* [10]

5. Native Database Options

a) --with-slice=value The number of database transactions an instance of a standalone native B-tree Mumps programs may perform on the database before relinquishing control. Default: 500

b) --with-cache=VAL native globals cache size [65537]

The only legal values for this parameter are:

9
17
33
65
129
257
513
1025
2049
4097
8193
16385
32769
65537
131073
262145
524289
1048577

c) --with-block=blksize native btree block size [8192]

The native Btree database consists of two files: the tree file (*key.dat*) containing the actual Btree and the data file (*data.dat*) containing stored data. The maximum size of the Btree file is dependent on the block size. The block sizes listed below each have a PAGE_SHIFT value and this ultimately determines the maximum file size as shown. The basic internal disk address is effectively 31 bits (signed 32 bit quantity) but, depending upon the block size, some number of bits at the low-order end are always zero. For example, if the block size is 1024, the final 10 bits of an address are always zeros. As only the significant 31 bits are stored, the true address is not 31 bits but 41 bits thus a file size of 2 terabytes is possible.

The only legal values for this parameter are:

1024

```
2048
4096
8192
16384
32768
65536
131072
262144
```

The block size determines the internal PAGE_SHIFT factor:

```
1024        →     PAGE_SHIFT 10
2048        →     PAGE_SHIFT 11
4096        →     PAGE_SHIFT 12
8192        →     PAGE_SHIFT 13
16384       →     PAGE_SHIFT 14
32768       →     PAGE_SHIFT 15
65536       →     PAGE_SHIFT 16
131072      →     PAGE_SHIFT 17
262144      →     PAGE_SHIFT 18
524288      →     PAGE_SHIFT 19
1048576     →     PAGE_SHIFT 20
2097152     →     PAGE_SHIFT 21

PAGE_SHIFT 10 corresponds to MBLOCK 1024 and a max Btree file size of 2 TB
PAGE_SHIFT 11 corresponds to MBLOCK 2048 and a max Btree file size of 4 TB
PAGE_SHIFT 12 corresponds to MBLOCK 4096 and a max Btree file size of 8 TB
PAGE_SHIFT 13 corresponds to MBLOCK 8192 and a max Btree file size of 16 TB
PAGE_SHIFT 14 corresponds to MBLOCK 16384 and a max Btree file size of 32 TB
PAGE_SHIFT 15 corresponds to MBLOCK 32768 and a max Btree file size of 64 TB
PAGE_SHIFT 16 corresponds to MBLOCK 65536 and a max Btree file size of 128 TB
```

The data file may grow to a max of 2**64 bytes for all settings.

d) --with-client — build native btree client data base code
e) --with-server-dir — native Btree server home directory [/etc/mumps]
f) --with-readonly — native database will be readonly – only applied to the native global array facility

6. --with-ibuf= — max size interpreted program [32000]
7. --with-strmax= — max internal string size [4096]
8. --with-locale=locale — locale information [en_US.UTF-8]
9. --with-terminate-on-error — halt interpretor on error [off]
10. --with-includes=DIR — to identify header dirs (Apple build only)
11. --with-libraries=DIR — to identify libs (Apple build only)
12. --with-float-bits=val — number of bits in floating point fractional part (72)
13. --with-float-digits=val — number of decimal digits to print in a floating point number (20)
14. --with-hardware-math — use hardware arithmetic facilities

2 Running a Mumps Program

2.1 Start the Global Array Server

Note: if you are using either the PosgreSQL, MySQL or native client /server option to store global arrays, you ***must*** start the PostgreSQL or MySQL server ***prior*** to attempting to access any global array.

2.2 Mumps CLI Interpreter

To run the command line interpreter from a terminal window, type:

```
mumps
```

Any Mumps commands you enter will be executed immediately. To exit the interpreter, type H[alt].

In interactive mode, you will be presented with a prompt (>). Any Mumps command may be typed for immediate execution (including a **goto** or **do** commands with a file name reference pointing to a file to be loaded and executed).

The keyboard *up arrow* and *down arrow* keys may be used to cycle through and display commands previously entered during this session.

A previously entered command my be re-executed by using the keyboard up arrow key to locate and display the command and then typing <enter>.

Input to the Mumps CLI follows GNU *readline* conventions.

2.2.1 Mumps CLI Special Commands

2.2.1.1 \globals

Lists the names of the global array tables and the number of columns in each (works only when using PostgreSQL or MySQL).

2.2.1.2 \halt \quit \h \q

Exit the Mumps CLI. The Mumps Halt command works as well.

2.2.1.3 \sys *cmd*

Executes *cmd* in a system shell then returns to the Mumps CLI. Example:

```
\sys ls -lh *
```

Expression substitution is permitted[13]:

```
> write $zsqlOutput
9910.tmp

> sql select * from doc where a1='1' limit 10;
> \sys cat &~$zsqlOutput~

1                       0
1     acetaldehyde      3
1     ribonuclease      6
1     alteration        7
1     catalytic         8
1     phosphate         10
1     ribonuclease      11
```

13 Note: The default contents of *$zsqlOutput* are the process id of the Mumps program followed by the *.tmp* extension.

```
1     react             12
1     acetaldehyde      13
1     cyanoborohydride  15
```

2.2.1.4 \mumps *pgm*

Runs the code in *pgm* as a Mumps program. The code need not begin with the *bash* interpreter comment. Expression substitution is permitted:

```
> \sys cat xxx.mps
 for i=%1:%2:%3 write i," "
 write !
> \mumps xxx.mps 1 2 10
1 3 5 7 9
> set x="1 2 10"
> set y="xxx.mps"
> \mumps &~y~ &~x~
1 3 5 7 9
```

2.2.1.5 \sql *sqlCommand*

Send *sqlCommand* to the PostgreSQL server and display any table output returned. Example:

```
> \sql select * from doc where a1='1' limit 10;
--------------------------
|1 |0                |   |
|1 |acetaldehyde     |3  |
|1 |ribonuclease     |6  |
|1 |alteration       |7  |
|1 |catalytic        |8  |
|1 |phosphate        |10 |
|1 |ribonuclease     |11 |
|1 |react            |12 |
|1 |acetaldehyde     |13 |
|1 |cyanoborohydride |15 |
--------------------------
```

Expression substitution is also possible:

```
> set i="doc",j=10
 > \sql select * from &~i~ where a1='1' limit &~j~;
--------------------------
|1 |0                |   |
|1 |acetaldehyde     |3  |
|1 |ribonuclease     |6  |
|1 |alteration       |7  |
|1 |catalytic        |8  |
|1 |phosphate        |10 |
|1 |ribonuclease     |11 |
|1 |react            |12 |
|1 |acetaldehyde     |13 |
|1 |cyanoborohydride |15 |
--------------------------
```

If your SQL command creates or drops a table, you **must** execute an **sql/f** command, withor without arguments, in order to update internal Mumps tables regarding the table(s) created or dropped.

2.3 Mumps Programs (scripts)

Mumps programs are ASCII files that can be created by any ASCII text editor. Do not use word processing editors that may embed hidden formatting characters into the text.

A script will normally have the following as their first line:

```
#!/usr/bin/mumps
```

The file extension of a Mumps program *.mps* is preferred but not required.

The Mumps source file must be made executable:

```
chmod u+x prog.mps
```

where *prog.mps* is the name of your mumps source file.

Example:

```
#!/usr/bin/mumps
      for i=1:1:10 do
      . write "Hello World ",i,!
      halt
```

You may execute the program by typing *prog.mps* to your terminal prompt. The program above will write *Hello World,* followed by a number ten times.

3 Relational Database Commands & Variables

3.1 Creating Global Array Relational Database Tables

As discussed above, you may enable storage of global arrays in either the MySQL or PostgreSQL relational database systems. While access to globals is slower than is the case with the native global array handler, the reliability and transaction processing capabilities offer many advantages.

In either MySQL or PostgreSQL, the globals will be stored in database tables that have the same names as the corresponding global arrays.

You should create the database tables before attempting to store globals in them or a default declaration will be done automatically. The default declaration will assume a predetermined number of columns (a *configure* option) which may not be correct.

Global arrays should be pre-declared with the *sql/f* command (see section 1.9.2.4 on page 14).

When a global array is stored in a table, the names of its columns are *a1, a2, a3, ...* The number of columns is set when the table is created, ordinarily by *sql/f*. However, you may use the SQL command *ALTER TABLE* to add or drop a column[14].

The builtin Mumps pseudo-variable *$zTabSize* (spelling is case insensitive) may be used to determine the number of columns in the most recently accessed table. *$zTabSize* is updated each time a global array is accessed.

The current table name in use is found in the system variable *$zTable* (spelling is case insensitive). This is the name of the most recently accessed global array table.

Neither *$zTabSize* nor *$zTable* are valid in a program until you have made at least one global array access.

Tables created by Mumps may be accessed outside of Mumps with standard SQL commands. The table name is the same as the global array name and the columns are *a1, a2, ... etc*. The data stored at a global array reference is always the last column.

3.2 Mumps Access to Relational Tables Not Created by Mumps

Tables or views created by programs other than Mumps may be accessed by Mumps if their column names and data types conform to Mumps or an appropriate *VIEW* for each is created that maps their actual column names to the column names used by Mumps (*a1, a2, a3, ...*). Mumps expects that the contents of each column be a text or character data type in order for all Mumps global array access functions to work correctly.

In your Mumps program, the non-Mumps table will be visible as a global array with the same name as the table. Note, however, the last column of the table will be interpreted as the value stored for the global array reference.

For example, the SQL commands in Figure 1 create and populate a small RDBMS table of temperature and dew point named *temps* with column names *city, temp,* and *dewpt*.

In order to make this table visible to Mumps, we map it to a view named *mtemps* that renames the columns to be compatible with Mumps. Note that the final column, *a4*, where Mumps normally stores values stored for a global array reference defined by the row, is defined as a constant empty string for each tuple. Also note that the data types of the columns are all a form of *varchar*.

```
drop view if exists mtemps;
drop table if exists temps;

create table temps ( city varchar(32), temp varchar(10), dewpt varchar(10));
```

14 If you use the *ALTER TABLE* command form a Mumps program, you should exit/restart the Mumps program before attempting to access the altered table so that Mumps may update its internal tables.

```
insert into temps values ('Boston', '32', '25');
insert into temps values ('Hyannis', '42', '32');
insert into temps values ('Norwood', '32', '12');
insert into temps values ('Quincy', '32', '24');
insert into temps values ('Waltham', '28', '23');

create view mtemps (a1,a2,a3,a4) as
      select city as a1, temp as a2, dewpt as a3, text '' as a4
      from temps;

select * from mtemps;
```

Figure 1 Creating a View

Figure 2 gives a mumps program to access the view from Figure 1.

```
#!/usr/bin/mumps

      write "mtemps",!

      for city=$order(^mtemps(city)) do
      . for temp=$order(^mtemps(city,temp)) do
      .. for dewpt=$order(^mtemps(city,temp,dewpt)) do
      ... write city,?15,temp,?20,dewpt,!

mtemps
Boston        32  25
Hyannis       42  32
Norwood       32  12
Quincy        32  24
Waltham       28  23
```

Figure 2 Mumps View Access

Alternatively, if the underlying table from which the view will be created contains numeric quantities, these can be cast as character strings in the view as shown in Figure3.

```
drop view if exists mtemps;
drop table if exists temps;

create table temps ( city varchar(32), temp int, dewpt int);

insert into temps values ('Boston', '32', '25');
insert into temps values ('Hyannis', '42', '32');
insert into temps values ('Norwood', '32', '12');
insert into temps values ('Quincy', '32', '24');
insert into temps values ('Waltham', '28', '23');

create view mtemps (a1,a2,a3,a4) as
      select city as a1, to_char(temp, '999') as a2,
            to_char(dewpt,'999') as a3, text '' as a4
      from temps;

select * from mtemps;
```

Figure 3 View with Data Conversion

Using a similar technique, most RDBMS tables can be viewed in Mumps. Note: MySQL and PostgreSQL at this time do not permit alteration of view values so the tables are *read-only*. Also note, the example in Figure 3 uses PostgreSQL data conversion functions. MySQL uses different functions.

3.3 SQL Commands in Mumps

If relational database storage of globals is enabled, the following commands are available in the Mumps interpreter. If the native database is in use, these are ignored.

3.3.1 sql *string*

The remainder of the line is passed to the SQL server. The line should contain a valid SQL command which is normally terminated by a semi-colon.

Any text of the form **&~*exp*~** will result in *exp* being evaluated with the result replacing **&~*exp*~**.

Mumps does not check the validity of the SQL command. The builtin Mumps variable *$zsql* will contain any messages returned by the RDBMS server or *'ok'* if there were none.

If you create or drop a table, you **must** inform Mumps by executing an **sql/f** command with no arguments. This causes Mumps to scan for the names of available tables and global arrays. Failure to do this may lead to erratic results. Alternatively, an **sql/f** command with arguments will also refresh the internal tables.

3.3.2 sql/c SQL Disconnect

The **sql/c** command disconnects from the SQL server. This is normally done automatically when a program terminates. No other text may appear on this line.

3.3.3 sql/f Format SQL Table

The command **sql/f** instructs the relational database server to create and initialize a table to store global arrays and to delete any previous contents the table may have had.

If the **sql/f** has no arguments, it refreshes the Mumps table that holds the list of known global arrays. This **must** be done if you **create** or **drop** a table using a direct SQL command. Alternatively, an **sql/f** command with arguments will also refresh the internal tables.

The *sql/f* command has two arguments:

1. a relational table name and

2. the number of columns in the table, including the column to store data.

The arguments must not contain embedded blanks. They are separated from one another by blanks.

The first argument is the name of the relational table to be created. It's name must conform to the naming requirements for a Mumps variable (no underscore characters, for example). It should be lower case as some relational database systems do not differentiate between upper and lower case table names.

The numeric argument may range between 1 and 20. It is the number of columns in the database that will contain stored data. This number is the maximum number of global array indices, minus one, for any global array reference to be stored in this table (the highest numbered column is for any stored values). Thus, if you will not have more than 10 indices in a table, this number should be 11. The first 10 will be used for global array indices and the 11^{th} for any stored data. The minimum value permitted is 2.

Examples:

```
sql/f labs 3
sql/f meds 5
```

Variables or expressions are not permitted.

You must include the name of the table to be created. Any existing table of the same name will be dropped (deleted) and a new one created. If you omit the second argument, the value currently in *$zTabSize* will be used to set the number of columns.

By default, the initial value of *$zTabSize* is 10 (this can be changed in *configure*).

3.3.4 Added Builtin SQL Variables

3.3.4.1 *$zsql*

Returns the SQL server error message for the most recent command or 'ok.'

3.3.4.2 *$zsqlOpen*

Returns true if a connection to the SQL server is open, false otherwise.

3.3.4.3 *$znative*

Returns true if globals are being stored in the native global array

3.3.4.4 *$zmysql*

Returns true if globals are being stored in MySQL

3.3.4.5 *$zpostgres*

Returns true if globals are being stored in PostgreSQL

3.3.4.6 *$ztable*

Returns a comma separated string. The portion prior to the comma is the the name of the most recently referenced table in which the Mumps globals are stored. The part after the comma is the maximum number of indices permitted in the table (same as *$ztabsize*).

3.3.4.7 *$ztabsize*

Returns the number of RDBMS columns available for global array indexes. May be set immediately prior to an *sql/f* command in which case the value in *$ztabsize* will be used to set the number of columns (range: 1 to 20) if a value is not explicitly given in the *sql/f* command.

3.3.4.8 $zsqlOutput

Contains the name of the file into which output from SQL commands will be written (see section 3.3.5). It may assigned a string value. This file name will be used by the SQL server to write output from *sql* commands. By default, the value in *$zaqlOutput* is the process id of the current program followed by the *.tmp* extension.

3.3.4.9 %globals()

Is an array containing the names of the Mumps tables in the relational database. Not currently available when using the native B-tree database. See also |*globals* section 2.2.1.1 on page 27. The following will list the available global array tables:

```
for i=$order(%globals(i)) write i,!
```

3.3.5 SQL Command Output

If you execute a SQL command from Mumps that has output (for example, a *SELECT* command), the output will be written to a sequential file. The name of the file in which the results will be put is contained in the builtin variable *$zsqlOutput*. By default, the name in this variable is the Mumps program's Linux process ID followed by a *.tmp* extension.'

For example:

```
set $zsqlOutput="results.dat"
sql select * from tab;
open 1:"rssults.dat,old"
```

```
...
```

You may set *$zsqlOutput* to any valid file name and output will subsequently be directed to that file. Each new SQL command will overwrite any prior contents of the file whose name is in *$zsqlOutput*. Files created by SQL output may be read by your Mumps program.

4 Mixing Mumps and SQL Code

In some cases it can be much faster to extract data from the relational database with a single SQL command rather that with a series of iterative Mumps loops where each Mumps global array reference becomes a SQL *SELECT* statement.

For example, suppose we have a global array named *^doc* which contains a *document-term* matrix, commonly used in information storage and retrieval (IS&R) experiments. The global array represents documents and the words that occur in them.

In this global array, each row is a document designated by a number and the columns are word stems. The value in an array element indexed by a document number and a word stem is a score indicating how important the particular word stem is in the document. For example:

```
^doc(411,"aortic")=32.32
^doc(411,"appearance")=4.39
^doc(411,"deteriorate")=5.26
^doc(411,"hemodynamic")=4.47
```

Figure 4, taken from an information retrieval system operating on medical documents, gives an example of how we might print such a matrix.

```
#!/usr/bin/mumps

 for d=$order(^doc(d)) do
 . write "document ",d
 . for w=$order(^doc(d,w)) do
 .. write " ",w,"(",^doc(d,w),")"
 . write !

...
document 411 aortic(32.32) appearance(4.39) deteriorate(5.26) hemodynamic(4.47)
document 446 antibiotic(4.92) bleede(4.39) bowel(4.16) cinedefecography(6.87)
document 458 added(4.39) antagonist(8.62) antagonize(11.54) atropine(32.69)
document 463 bile(8.46) bile-duct(17) biochemical(8.32) canalicular(6.18)
...
```

Figure 4 Mixing Mumps and SQL

In IS&R research we need a table of all the words used by all documents and, for each word, the number of documents in which it occurs.

We could do this in Mumps with the code shown in Figure 5 where we calculate the vector *^df* which gives, for each word, the number of documents in which the word occurs.

```
  for d=$order(^doc(d)) do
 . write "document ",d
 . for w=$order(^doc(d,w)) do
 .. if '$data(^df(w)) set ^df(w)=0
 .. else  set ^df(w)=^df(w)+1
```

Figure 5 Expensive Iterative Mumps Code

However, in a relational database, this calculation generates a massive number of SQL queries. Each global array reference is a query! We could, however, create the same table with the SQL command shown in Figure 6[15].

```
 sql drop table if exists df;

 if $zsql'="ok" write "Error dropping df ",$zsql,! halt
```

15 The \ character causes line continuation beginning at the next non-blank or non-tab character.

```
sql create table df as \
     select a2 as a1, \
           trim(both from to_char(count(*),'9999999999')) as a2 from \
                 doc group by a2;

if $zsql'="ok" write "Error creating df ",$zsql,! halt
```

Figure 6 SQL Equivalent

In Figure 6, the long *create table* command has been wrapped onto multiple lines for ease of reading.

In the SQL *create table* command we build the global array *^df* consisting of a column of words (*a1*) *and counts (*a2*)* by selecting from *^doc* words *a2* from *^doc* and the count of the number of time they occur.

The code:

```
   trim(both from to_char(count(*),'9999999999')) as a2
```

is how PostgreSQL[16] converts the numeric result from *count(*)* to character string, the result of which is labeled *a2 (*Mumps prefers character data). The word, which is *a2* in *^doc,* becomes *a1* in *^df.*

By grouping the rows of *^doc* by word (*a2*), we get a set of groups by word. The *count* function counts the number of elements in these groups and the result is a word and its count.

While this appears complicated, it isn't. It's fairly standard SQL code and efficient. Consequently, the construction of *^df* is enormously quicker. The code from Figure 6 may be inserted into a larger Mumps program as shown in Figure 7. A sample of the output is shown[17].

```
#!/usr/bin/mumps

 for d=$order(^doc(d)) do
 . write "document ",d
 . for w=$order(^doc(d,w)) do
 .. write " ",w,"(",^doc(d,w),")"
 . write !

 sql drop table if exists df;

 if $zsql'="ok" write "Error dropping df ",$zsql,! halt

 sql create table df as \
      select a2 as a1, \
           trim(both from to_char(count(*),'9999999999')) as a2 from \
                 doc group by a2;

 if $zsql'="ok" write "Error creating df ",$zsql,! halt

 for w=$order(^df(w)) do
 . write w,?20,^df(w),!


.
.
.
document 411 aortic(32.32) appearance(4.39) deteriorate(5.26) hemodynamic(4.47)
document 446 antibiotic(4.92) bleede(4.39) bowel(4.16) cinedefecography(6.87)
document 458 added(4.39) antagonist(8.62) antagonize(11.54) atropine(32.69)
```

16 MySQL has a simpler function to achieve the same result.

17 The *for d=$order(...)* format is peculiar to this version of Mumps. It was not part of any of the legacy standards.

```
document 463 bile(8.46) bile-duct(17) biochemical(8.32) canalicular(6.18)
.
.
.
acetaldehyde       6
acetaldehyde-rnase 2
acetate            6
acetate-salt       2
acetic             2
acetylcholine      9
acetyltransferase  2
ach-induce         4
achlorhydric       2
achromatic         5
achromaticity      2
achromatic-target  1
.
.
.
```

Figure 7 Merged Mumps/SQL Code

Note also, the program in Figure 7 could have also been written as shown in Figure 8. In this version, a SQL *select* command is used to generate a file of output which is read and formatted by mumps. It is faster in that it involves fewer transactions with the server. Note that, by default, columns in the SQL output are separated by *<tab>* characters which appear as the *$char(9)* delimiter codes in the *$piece()* function.

```
#!/usr/bin/mumps

 for d=$order(^doc(d)) do
 . write "document ",d
 . for w=$order(^doc(d,w)) do
 .. write " ",w,"(",^doc(d,w),")"
 . write !

 sql drop table if exists df;

 if $zsql'="ok" write "Error dropping df ",$zsql,! halt

 sql create table df as \
      select a2 as a1, \
            trim(both from to_char(count(*),'9999999999')) as a2 from \
                  doc group by a2;

 if $zsql'="ok" write "Error creating df ",$zsql,! halt

 set $zsqlOutput="DocFreq.txt"

 sql select * from df order by a1;

 open 1:"DocFreq.txt,old"

 for  do
 . use 1 read line if '$test break
 . use 5
 . write $piece(line,$char(9),1),?20,$piece(line,$char(9),2),!
```

Figure 8 Alternative Mumps/SQL Code

5 Implementation Notes

5.1 GOTO Command

If you use a **goto** command, all **do** command pending returns are canceled. That is if you invoke a section of code by means of a **do** and the section of code executes a **goto** command, the return to the line the **do** was on is canceled as well as any other pending returns.

5.2 Notes on Arithmetic Precision

See section 1.12 on page 23 for additional details.

5.2.1 $fnumber()

The builtin function **$fnumber()** only works on numbers that can be represented in a 64 bit floating point variable.

5.2.2 Exponential format numbers

All numbers represented in exponential format are treated as floating point numbers. If exponential format constants are used in expressions, they must be enclosed in quotes:

```
set i="1.23e3"*5
```

5.2.3 Arithmetic Precision

If found, Mumps will use the GNU *bignum* integer and MPFR floating point packages (this can be disabled by a *configure* option).

5.2.3.1 Floating Point Precision

When using extended precision MPFR numbers, floating point values have a default fractional precision of 72 bits. This can be changed with the *--with-float-bits=val* configure option. The maximum number of printed decimal digits is, by default, 20. This can be changed with the *--with-float-digits=val* configure option. The number of meaningful decimal digits that can be printed depends upon the number of bits in the fractional part of the floating point number. More bits mean more decimal digits can be printed.

If MPFR is not present, standard hardware *double* precision is used.

5.2.3.2 Integer Precision

There is no effective limit to integer precision except string length and memory when the extended precision *bignum* package is in use. Otherwise, precision is the same as the hardware *long.*

5.2.3.3 Performance

Extended precision arithmetic results in slower performance. The amount is dependent on how much arithmetic a program does, whether it is mainly integer or floating point (floating point is slower), and, in the case of fixed length numbers, how large the numbers are. Larger numbers result in slower computations.

5.2.4 Rounding

The *$justify()* function is useful to round lengthy repeating decimal floating point numbers to a more reasonable value.

5.3 New Command

The **new** command functions differently than in the 1995 standard. The following details its behavior.

5.3.1 Runtime Symbol Table

The **new** command controls the internal run time symbol table. Upon entering a block by means of a **do** command, a new layer of the symbol table is created. Upon exit, the layer is discarded and the previous layer becomes the current layer.

When a program begins, an initial or base layer is created in the symbol table. In the absence of any **new** commands, newly created variables are stored at this base or initial layer.

When a variable is retrieved, all layers are searched beginning with the most recently created layer and progressing through to older layers until the initial layer is reached.

In the absence of any **new** commands, only the initial or base layer will contain variables.

5.3.2 Forms of the New Command

There are three forms of the **new** command based on the arguments provided. The first has no arguments, the second has a list of arguments consisting of variable names separated from one another by commas, and, finally, the third has an argument consisting of a parenthesized comma separated list of variable names. For example:

```
new
new a,b,c
new (a,b,c)
```

5.3.2.1 New Command with No Arguments

A **new** command with no arguments cause the system to copy all variables from all layers to the current layer.

Until the current block is exited, all access to any variable known at the time of the **new** command will access the copy of the variable, not the original. Upon exit from the block, the copies are deleted[18].

Any variable created whose name was not known when the **new** command was executed, will be created and stored at the lowest base layer of the symbol table and, consequently, not deleted upon exit from the block that contained the **new** command.

If a **new** command is executed in a block that invokes a block which itself executes a **new** command, the **new** command in the second block makes of copy of the invoking block's variables along with any variables created by the invoking block after executing its **new** command. If, in the symbol table stack, a vriable appears at several layers, only the most recent version will be copied.

An example is given in Figure 9. In this example, variables *i, j,* and *k* are created at the beginning of the program. The function *test1* is then called.

Initially, in *test1*, the variables have the same values that they did in the main function. The variable *i* is changed. The **new** command is executed and a copy of all the variable currently known (*i,j,k*) is made to the current layer. The values of *i, j,* and *k* are altered the function *test2* is called.

The values of the variables on entry to *test2* are the same as they were in *test1*. Anothe **new** command is executed making another copy of the variables. These are altered and a new variable, *y*, not previously known at any level (and thus stored at the base level) is created. Return is made to *test1.*

In *test1* the values of the variable are printed and it can be seen that they have reverted to the values they had prior to entering *test2*. Return is made to the main function.

In the main function the variables have reverted to the values they had prior to the invocation of *test1* with the exception of *i* which was altered in *test1* prior to execution of the **new** command. It retains the value it received in *test1*.

Note also that the variable *y* now exists at the main function level since, when it was created in *test1*, it was not in the group of variables copied to the symbol table level for *test1*. Thus, it was created at the base level of the symbol table.

18 A block is any sequence of code entered as a result of a **do** command.

However, when *y* was altered in *test2*, only the copy made by the **new** command in *test2* was altered, not the original.

```
#!/usr/bin/mumps
        set i=10
        set j=20
        set k=30
        do test1
        write "Main: expect 100 20 30 50: ",i," ",j," ",k," ",y,!
        halt

test1 write "test1: expect 10 20 30: ",i," ",j," ",k,!
      set i=100
      new
      set i=11,j=22,k=33,y=50
      do test2
      write "test1: expect 11 22 33 50 : ",i," ",j," ",k," ",y,!
      quit

test2 write "test2: expect 11 22 33 50: ",i," ",j," ",k," ",y,!
      new
      set i=12,j=23,k=34,y=55
      write "test2: expect 12 23 34 55 : ",i," ",j," ",k," ",y,!
      quit

root@AMD6 validate new01.mps

test1: expect 10 20 30: 10 20 30
test2: expect 11 22 33 50: 11 22 33 50
test2: expect 12 23 34 55 : 12 23 34 55
test1: expect 11 22 33 50 : 11 22 33 50
Main: expect 100 20 30 50: 100 20 30 50
```

Figure 9 **new** Command without Arguments

5.3.2.2 New Command with Arguments

There are two forms of the **new** command that take arguments.

The first has a list of arguments consisting of variable names separated from one another by commas:

```
new a,b,c
```

The second has an argument consisting of a parenthesized, comma separated list of variable names:

```
new (a,b,c)
```

If a variable is named in the list that does not exist, it is created in the current symbol table layer with a value of the empty string.

5.3.2.2.1New Command with Comma List of Variable Names

If the **new** command argument is a list of one or more variable names, it means that the variables listed will be copied to the current symbol table level and, eventually, discarded when the current block is exited[19].

If a variable whose name appears in the list exists at several layers in the symbol table stack, only the most recent will be copied.

19 A block is any sequence of code entered as a result of a **do** command.

Any reference to any variable not in the argument list will be satisfied by searching through the symbol table stack for the most recent instance of it. See Figure 10.

If a variable is mentioned in the argument list that does not exist, it is ignored.

```
#!/usr/bin/mumps
        set i=10
        set j=20
        set k=30
        do test1
        write "Main: expect 100 20 30 50: ",i," ",j," ",k," ",y,!
        halt

test1 write "test1: expect 10 20 30: ",i," ",j," ",k,!
      set i=100
      new i,j
      set i=11,j=22,k=33,y=50
      do test2
      write "test1: expect 11 23 34 55 : ",i," ",j," ",k," ",y,!
      quit

test2 write "test2: expect 11 22 33 50: ",i," ",j," ",k," ",y,!
      new i
      set i=12,j=23,k=34,y=55
      write "test2: expect 12 23 34 55 : ",i," ",j," ",k," ",y,!
      quit

root@AMD6 validate # new02.mps

test1: expect 10 20 30: 10 20 30
test2: expect 11 22 33 50: 11 22 33 50
test2: expect 12 23 34 55 : 12 23 34 55
test1: expect 11 23 34 55 : 11 23 34 55
Main: expect 100 20 30 50: 100 20 34 55
```

Figure 10 **new** Command with Comma List

5.3.2.2.2 New Command with Parenthesized List of Variable Names

If the **new** command argument list consists of a parenthesized list of one or more variable names, it means to make a copy of the most recent versions of all known variables except for the variable named in the list. This is similar to the no-argument version except the one or more variables known at the time of command execution will not be copied to the current symbol table layer.

When the block containing the **new** command is exited, the copies of the variables are discarded but any changes to this variables given in the argument list are not[20].

See Figure 11.

```
#!/usr/bin/mumps
        set i=10
        set j=20
        set k=30
        do test1
        write "Main: expect 11 22 30 50: ",i," ",j," ",k," ",y,!
        halt

test1 write "test1: expect 10 20 30: ",i," ",j," ",k,!
      new (i,j)
        set i=11,j=22,k=33,y=50
```

20 Note: if one or more of the variables in the argument list are themselves copies from a lower layer but not the base layer, they will eventually be discarded.

```
      do test2
      write "test1: expect 11 23 34 55 : ",i," ",j," ",k," ",y,!
        quit

test2 write "test2: expect 11 22 33 50: ",i," ",j," ",k," ",y,!
      new i
      set i=12,j=23,k=34,y=55
      write "test2: expect 12 23 34 55 : ",i," ",j," ",k," ",y,!
      quit

root@AMD6 validate # new03.mps

test1: expect 10 20 30: 10 20 30
test2: expect 11 22 33 50: 11 22 33 50
test2: expect 12 23 34 55 : 12 23 34 55
test1: expect 11 23 34 55 : 11
```

Figure 11 **new** Command with Parenthesized List

5.4 Kill Command

The **kill** command operates only on the current symbol table level.

5.5 Lock Command with PostgreSQL or MySQL

Locks are not needed if using the PostgreSQL or MySQL for global array storage as SQL transaction commands can achieve the same effect. When using PostgreSQL or MySQL for the backend global array stores, the Lock should not be used. Instead, use the more modern native SQL transaction processing commands (*BEGIN, COMMIT, ROLLBACK*, etc.) to achieve the same effect with far greater integrity.

5.6 Lock command in client/server mode

In native B-tree mode, the Lock command creates a file named *Mumps.Locks* in */tmp* where the lock information for the system is stored. If this file becomes corrupted due to abnormal terminations, it should be deleted. It will be rebuilt as needed.

5.7 Line Continuation

A line may be continued by placing a backslash at its end. The next line is appended beginning with the first non-blank or non-tab character. Note: this means that a blank must be on the prior line. Example:

```
 sql create table df as select a2 as a1, trim(both from \
      to_char(count(*),'9999999999')) \
      as a2 from doc group by a2;
```

5.8 Naked indicator

This version of Mumps does not support the naked indicator. The naked indicator has no place in a modern or even semi-modern programming language. It was originally included in early versions of Mumps because of the inefficient binary mapping of an n-way tree which was used at the time to store the global arrays. The naked indicator was a short-hand to the interpreter to allow it to search for a global without stating at the top of the tree each time thus resulting in faster access. That is no longer the case with modern B-tree based access methods. Another issue is the perceived ambiguity of determining what exactly the naked indicator is after certain Mumps operations. Unfortunately, some legacy applications use it. These should be re-written.

5.9 Job command

The **JOB** command results in a C/C++ *fork()* function to be executed thus creating a child process. The child process will attempt to execute the argument to the **JOB** command. The **JOB** command may be used in native B-tree user mode but only one process may access the globals. In native client server mode, this restriction is not in effect. For PostgreSQL and MySQL, the child process should create a new connection.

The child process must end with a **HALT** command or the child process will hang.

5.10 File Names Containing Directory Information

When invoking a file name containing directory information (forward slash in Linux and backslash in DOS) with the **DO** or **GOTO** commands, the file name **must** be enclosed in quotes. For example:

```
set x="""^/home/user/xxx.mps""" goto @y
goto @"""^/home/user/xxx.mps"""
```

Note the extra quotes. These are required.

5.11 File Names

File names should conform to variable naming conventions except that the first character of a file name may not be the per cent sign (%). The first character must be alphabetic. File names may only contain letters, digits and the per cent sign.

5.12 Array Index Collating Sequence

Array index collating sequences for both global and local array is ASCII. That is, for the *$query()* and *$order()* functions, all array indices will be presented in the same order as ASCII strings. Thus, in an array with 15 elements whose indices range from 1 to 15, the indices will be presented as:

```
1 10 11 12 13 14 15 2 3 4 5 6 7 8 9
```

Other versions of Mumps may present numeric indices in numeric order. This, however, leads to considerable inefficiencies in the data base.

You may achieve numeric ordering by storing the indices padded to left with blanks such as:

```
for i=1:1:15 set ^a($justify(i,8))=i
set i="" for  set i=$order(^a(i)) quit:i='' write +i," "
```

the indices will now be presented as:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Note the the +i in the **write** command has the effect of converting the string to a number with no leading blanks.

5.13 Subroutine & Function Calls

Subroutines and functions may be performed in several ways as shown in Figure 12. Values returned from functions invoked by a **do** command are ignored. In standard Mumps, the *$$* form is used only with function invocations.

Caution: be certain to include a **halt** or other exit in your program prior to any functions. If the **halt** is not present, function code will be entered and any passed variables will be undefined.

```
#!/usr/bin/mumps
# calls.mps

     set i=10
     do fcn(i)
     do fcn(5)
     do $$fcn(i)
     do $$fcn(5)
     set k=$$fcn(5)
     write "returned k=",k,!

     set i=10
     do fcn^ext.mps(i)
     do fcn^ext.mps(5)
     do $$fcn^ext.mps(i)
```

```
      do $$fcn^ext.mps(5)
      set k=$$fcn^ext.mps(5)
      write "returned k=",k,!

      do fcn^ext1.mps
      do fcn^ext1.mps
      do $$fcn^ext1.mps
      do $$fcn^ext1.mps
      set k=$$fcn^ext1.mps
      write "returned k=",k,!

      halt

fcn(x) write "in fcn(x) value passed is ",x,!
       quit x

------------------------------

#!/usr/bin/mumps
# ext.mps

fcn(x) write "in fcn(x) value passed is ",x,!
       quit x

------------------------------

#!/usr/bin/mumps
# ext1.mps

fcn    write "in fcn ext1.mps",!
       set x=22
       quit x
------------------------------

output results:

in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 5
returned k=5
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 10
in fcn(x) value passed is 5
in fcn(x) value passed is 5
returned k=5
in fcn ext1.mps
in fcn ext1.mps
in fcn ext1.mps
in fcn ext1.mps
in fcn ext1.mps
returned k=22
```

Figure 12 Subroutine/Function Calls

5.14 $Fnumber() Function

The **$fnumber()** is implemented via the C function *strfmon()* which provides much greater flexibility when dealing with differing locales and, especially, currencies. The default locale is *en_US.UTF-8* but this can be set with the *configure* option:

```
--with-locale=location-information
```

You may use **$fnumber()** with the legacy Mumps parameters or use it with a pattern paramter designed for *strfmon()*.

If you use the *strfmon()* parameter opetion, the function takes two arguments. The first must be a number consisting of only numeric characters. The second is a character string conforming to a *strfmon()* pattern but preceded by an asterisk to distinguish the pattern from those used by the legacy Mumps function of the same name. The *strfmon()* function is well documented but here are some examples:

```
set x=12345.6789
write $fn(x,”*%!n”)     ==> 12,345.68
write $fn(x,”*%n”)      ==> $12,345.68
write $fn(x,”*%i”)      ==> USD 12,345.68
write $fn(x,”*%n3”)     ==> $12,345.683
write $fn(x,"*%20n")    ==>           $12,345.68
```

5.15 $select() Function

All arguments of the **$select()** function are evaluated.

5.16 Compiling Large Programs

When compiling[21] large programs, especially if MySQL is enabled, there may be a warning about *variable tracking* from the gcc/g++ compiler. You may ignore this.

5.17 Embedded Expressions

In several extended Mumps commands, the figure *&~exp~* may appear. The expression *exp* is evaluated and the result replaces the figure. For example:

```
set x=”ls -lh”
shell &~x~

set x= “select * from abc;”
sql &~x~
```

5.18 Functions

This is the form of subroutine was originally used in Mumps. There are no parameters passed to the subroutine and the subroutine shares the same *namespace* as the calling program hence, as seen in the example in Figure 13, the values of the variables *i*, *j*, and *k* are accessible to the subroutine and any changes to them are available in the calling program.

Variables created in the subroutine in the normal manner by a **set** or **read** command, unless the subject of a **kill** command, are available to the calling routine.

Variables created in the subroutine as a result of a **new** command are destroyed upon return and are not available to the calling routine.

```
        zmain
        set i=10
        set j=20
        set k=30
        write "main program: ",i," ",j," ",k,!
        do test
        write "main program: ",i," ",j," ",k,!
        write "main program x=",x,!
        write "main program $data(y)=",$data(y),!
        halt

test
        write "sub-program: ",i," ",j," ",k,!
```

21 Using the compiler is not presently recommended.

```
        set i=11
        set j=22
        set k=33
        set x=22
        new y
        set y=33
        quit

which produces the following output:

main program: 10 20 30
sub-program: 10 20 30
main program: 11 22 33
main program x=22
main program $data(y)=0
                          Figure 13 Inline Functions
```

5.18.1 Call by Value

This form of subroutine call was introduced later in the evolution of Mumps. It permits parameters to be passed to the subroutine but the subroutine maintains a separate name space for values passed to it as parameters. Variables from the calling program are visible to the called program. Variables created by the called program become available to the calling program upon return (except if the are **kill**ed prior to return or created by a **new** command). and variables created in the called program are deallocated upon return and are thus not visible to the calling program. Changes to parameters passed to the called program do not change the corresponding arguments in the calling program.

```
        zmain
        set i=10
        set j=20
        set k=30
        write "main program: ",i," ",j," ",k,!
        do test(i,j,k)
        write "main program: ",i," ",j," ",k,!
        halt

test(a,b,c)
        write "sub-program: ",a," ",b," ",c,!
        set a=11
        set b=22
        set c=33
        quit

which produces the following output:

main program: 10 20 30
sub-program: 10 20 30
main program: 10 20 30

                      Figure 14 Call by Value Functions
```

5.18.2 Call by Reference.

Same as the above but 'call be reference' permitted. That is, changes to parameters made by the called program cause changes to the corresponding arguments in the calling program. Note the "." in front of the variables in the 'do' command that are to be passed by reference. Both call by reference and call by value arguments may be mixed in the same 'do' statement.

```
#!/usr/bin/mumps
        zmain
```

```
        set i=10
        set j=20
        set k=30
        write "main program: ",i," ",j," ",k,!
        do test(.i,.j,.k)
        write "main program: ",i," ",j," ",k,!
        halt

test(a,b,c)
        write "sub-program: ",a," ",b," ",c,!
        set a=11
        set b=22
        set c=33
        quit

which produces the following output:

main program: 10 20 30
sub-program: 10 20 30
main program: 11 22 33
```

Figure 15 Call by Reference Functions

In each of the examples, the subroutine and calling program are actually part of the same C++ function. In effect, subroutines of the type shown above as similar to the old Basic ***gosub*** facility. Functions such as shown above may also return values:

An example recursive factorial computation is shown in Figure 16.

```
#!/usr/bin/mumps
      zmain
      set i=$$factorial(5)
      write "factorial=",i,!
      halt

factorial(a)
      write "sub-program: a=",a,!
      if a<2 quit 1
      set b=$$factorial(a-1)
      write "a=",a," b=",b,!
      quit a*b

sub-program: a=5
sub-program: a=4
sub-program: a=3
sub-program: a=2
sub-program: a=1
a=2 b=1
a=3 b=2
a=4 b=6
a=5 b=24
factorial=120
```

Figure 16 Function Return Values

6 Shell Command

6.1 shell

6.2 shell/g

6.3 shell/p

The **shell** command passes the remainder of the line to a shell for execution (**sh** in Linux). Shell output will appear on **stdout**. The command sets **$test** to false if the *fork()* fails, true otherwise.

This command is not presently available in the DOS version.

The **shell/p** form passes the remainder of the line to a shell for execution but opens a pipe **from** the shell **to** Mumps unit number 6. All **stdout** output from the shell is directed to unit number 6 and can be read with any of the input commands or functions in association with the **use** command.

The **shell/g** form passes the remainder of the line to a shell for execution (**sh** in Linux) and opens a pipe **from** the Mumps program **to** the shell as Mumps unit number 6. Data **written** to this unit becomes **stdin** to the shell. Output from the shell is written to **stdout**. Remember to **close** unit number 6 to signal end-of-file to the shell.

With no qualifier, the **shell** command passes the remainder of the command line to a shell. Input or output from the shell come from or go to **stdin** or **stdout**, respectively.

In all cases, the remainder of the command line is scanned for **&~...~** expressions. The expression between **&~** and **~** is evaluated and the result replaces the **&~...~** expression.

For example:

```
shell sort dictionary.tmp | uniq -c | sort -nr > dictionary.s
```

The Linux shell created will do the following:

1. The file *dictionary.tmp,* a collection of words, will be sorted by **sort** and the output piped to **uniq**
2. **uniq** counts duplicate entries and pipes its output consisting of a count and a word to **sort**
3. **sort** sorts the result numerically by number of duplicates in reverse order and writes its output to *dictionary.s*.

```
1  shell/p sort dictionary.tmp | uniq -c | sort -nr
2  open 1:"dictionary.s,new"
3  for  do
4  . use 6
5  . read line
6  . if '$test break
7  . use 1
8  . write line,!
9  close 1
```

Figure 17 Shell Command Example

The above does the same but the output will be presented to Mumps unit 6 which reads and writes the result to the file named dictionary.s

7 Added Commands

7.1 Database *expr*

The **database** command may be used to set the name of the files to be used to store the native global arrays. The expression will be evaluated and the resulting name will become the name, suffixed *.key* and *.dat,* of the files in which the native global arrays are stored. The expression may contain directory information. For example:

```
database "/home/user/data/mumps"
```

will cause the system to access files:

```
/home/user/data/mumps.key
/home/user/data/mumps.dat
```

for the global array tree and data files. If directory information is omitted, the files will be in the current directory.

This command *must* be issued prior to any attempt to access the global arrays. It only works with the native B-tree database option.

7.2 Zhalt return_code

The **zhalt** command will terminate the current program with a return error code given by its argument. Example:

```
   if a=0 zhalt 99
```

The value of **$?** in the BASH environment will be 99.

8 Z Functions and System Variables

$zfunctions are extensions added by the implementor and not covered by the standard. Thus, many if not all of the following M2 extensions may not be supported or supported differently in other implementations. Likewise, there are implementer defined system variables which may be queried and, in some cases, set.

M2 implementation note: you may add new **$z** functions by modifying the function **zfcn()** located in the source file *bifs.cpp.in*

8.1 System Variables

8.1.1 $ztable

The value in *$ztable* is the name of the current database table if you are using PostgreSQL or MySQL to store the global arrays. It has no meaning if you are using the native Btree. The default value will be *mumps* unless this was changed during the *configure* step or changed during execution.

You may set the name of the database table in use by setting *$ztable*. All database references will take place in the table you set *$ztable* to until you change it again or the program terminates.

When you start a Mumps program, *$ztable* reverts to the value set by *configure.*

The string returned by *$ztable* consists of two parts separated by a comma. The first part is the name of the table and the second part is the number of columns available for global array indices. The number is indices is two less than the actual number of columns as one column is reserved for the global array name named *gbl*, and one column for the value stored at the global reference named *ax* where the value of *x* is one greater that the number of indices.

If you change the default table, you need to insure that it exists and is properly defined for use my Mumps. The *sql/f* command will create and initialize a new table or re-initialize to empty an existing table. Warning messages may appear the first time you create a table.

8.1.2 $zTabSize

The maximum number of indices permitted in a global array reference. When queried, this variable returns the current setting. It may be set (maximum of 20). If you set it, you ***must*** initialize the global array before using it or errors will result.

8.1.3 $zProgram

Returns a string with the name of the currently executing program.

8.2 Math Functions

The following C/C++ math functions are available in M2. Their arguments and return values are the same as the correspondingly named C++ functions.

8.2.1 $zabs(arg) absolute value

Function returns the absolute value of its numeric argument.

8.2.2 $zacos(arg) arc cosine

Computes the inverse cosine (arc cosine) of the input value. Arguments must be in the range -1 to 1.

8.2.3 $zasin(arg) Arc sine

Computes the inverse sine (arc sine) of the argument **arg**. Arguments must be in the range -1 to 1.

8.2.4 $atan(arg) Arc tangent

Computes the inverse tangent (arc tangent) of the input value.

8.2.5 $zcos(arg) Cosine

Computes the cosine of the argument **arg**. Angles are specified in radians.

8.2.6 $zexp(arg) Exponential

Calculates the exponential of **arg**, that is, ***e*** raised to the power *arg* (where ***e*** is the base of the natural system of logarithms, approximately 2.71828).

8.2.7 $zexp2(arg) Exponential base 2

Calculates 2 raised to the power *arg*.

8.2.8 $zexp10(arg) Exponential base 10

Calculates 10 raised to the power *arg*.

8.2.9 $zlog(arg) Natural log

Returns the natural logarithm of **arg**, that is, its logarithm base ***e*** (where e is the base of the natural system of logarithms, 2.71828...).

8.2.10 $zlog2(arg) Base 2 log

Returns the base 2 logarithm of ***arg***.

8.2.11 $zlog10(arg) Base 10 log

Returns the base 10 logarithm of **arg**.

8.2.12 $zpow(arg1,arg2) Power function

Calculates **arg1** raised to the exponent **arg2**.

8.2.13 $zsqrt(*arg*) Square root

Function returns the square root of its numeric argument.

8.2.14 $zsin(arg) Sine function

Computes the sine of the argument **arg**. Angles are specified in radians.

8.2.15 $ztan(arg) Tangent function

Computes the tangent of **arg**.

8.3 Date functions

8.3.1 $zdate(or $zd) formatted date string

Function returns the system date and time in standard system printable format. This includes: day of week, month, day of month, time (hour:minute:second), and year (4 digits).

8.3.2 $zd1 numeric internal date

Returns the number of seconds since January 1, 1970 - a standard used in Linux. This number may be used to accurately correlate events.

8.3.3 $zd2(***InternalDate***) date conversion

Translates the Linux time from $ZD1 into standard system printable format. The argument is a Linux format time value.

8.3.4 $zd3(***Year,Month,Day***) Julian date

Returns the day of the year (Julian date) for the Gregorian date argument.

8.3.5 $zd4(*Year,DayOfYear*) Julian to Gregorian

Returns the Gregorian date for the Julian date argument.

8.3.6 $zd5(*Year, Month, Day*) comma listed date

Returns a string consisting of the year, a comma, the day of year, and the number of days since Sunday (Monday is 1).

8.3.7 $zd6 hour:minute

Returns a string consisting of the hour, a colon, and the minute.

8.3.8 $zd7 hyphenated date

Returns a string consisting of the year, hyphen, month, hyphen, and day of month. If an argument is given in the form of the number of seconds since Jan 1, 1970, the result returned will reflect the argument date.

8.3.9 $zd8 hyphenated date with time

Returns a string consisting of the year, hyphen, month, hyphen, and day of month, comma, and time in HH:MM format. If an argument is given in the form of the number of seconds since Jan 1, 1970, the result returned will reflect the argument date.

8.4 Special Purpose Functions

The following special purpose functions are available:

8.4.1 $zb(arg) remove blanks

Function returns a string in which all leading blanks have been removed and all multiple blanks have been replaced by single blanks. See also **$zNoBlanks()**. Figure 18 gives examples.

```
1 #!/usr/bin/mumps
2  set a="   abc   xyz     123    "
3  write $zb(a),"***",!

output:

abc xyz 123 ***
```

Figure 18 $Zb() Examples

8.4.2 $zchdir(directory_path) change directory

Function changes the current directory to the path specified. If the operation succeeds, a zero is returned. If it fails, -1 is returned.

8.4.3 $zCurrentFile Current Mumps File

Returns the name of the currently executing Mumps program file (if any) ar blank.

8.4.4 $zdump[(filename)] dump global arrays

Function dumps the globals to a sequential ASCII file in the current directory. If an argument is given, it is taken as the name of the file to which the globals will be written. If the argument is omitted, a file name is constructed from the system date of the form **number.dmp** where **number** is the value of the C++ **time()** function at the time of the dump.

The dump file is a pure ASCII text file. Each entry in the global array is represented by two lines. The first line is the global array reference and the second line is the store value. In the global array reference, parentheses and commas are replaced by the "~" character. Thus, if you wish to use this facility, you may not include the "~" character in a global array index.

The function **$zrestore()** reloads the global arrays from a dump file (see below).

$zdump and **$zrestore** do not work when PostgreSQL is used for the global array store.

8.4.5 **$zrestore[(arg)]** restore globals

Function restores the globals from a dump file produced by **$zdump**. If an argument is given, it is taken as the name of the dump file otherwise, the default name **dump** is used.

$zdump and **$zrestore** do not work when PostgreSQL is used for the global array store.

8.4.6 **$zfile(*arg*)** file exists test

Function returns a zero or one indicating if the file given as the argument exists.

8.4.7 **$zflush** flush Btree buffers

Function flushes all modified native global array handler buffers to disk. The function should only be used with the native globals. After flushing, all updates to the btree file system have been committed. In cases where the internal buffers are very large, this function may take several seconds to execute. The function returns the empty string. Flushing the buffers is a precaution against system failure which would otherwise result in corruption of the global arrays.

8.4.8 **$zgetenv(*arg*)** get environment variable

Returns the contents of the environment variable specified as *arg* or the empty string if the variable is not found.

8.4.9 **$zhtml(*arg*)** encode HTML string

Function encodes its argument in the form necessary to be a cgi-bin parameter. That is, alphabetics remain unchanged, blanks become plus signs and all other characters become hexadecimal values, preceded by a percent sign.

8.4.10 **$zhit** global array cache hit ratio

Function calculates and returns the native global array cache hit ratio. This number ranges between zero and one. A value of one indicates all requests were satisfied from the cache while a value of zero indicates no requests were satisfied from the cache. Calling this function resets the hit ratio to zero. A higher value for the hit ratio indicates better database performance.

8.4.11 **$zlower(string)** convert to lower case

Function returns the input string with alphabetics converted to lower case.

8.4.12 **$znormal(*arg1*[,arg2])** word normalization

Function converts the word passed as argument 1 to lower case and removes any embedded punctuation. If a second argument is given, the word is truncated to the length specified by this argument. If no second argument is given, words are truncated to 25 characters if their length exceeds 25 characters.

8.4.13 **$zNoBlanks(arg*)*** remove all blanks

Returns ***arg*** with all blanks removed. See also: **$zb**.

8.4.14 **$zpad(arg1,arg2)** left justify with padding

Function left justifies the first argument in a string whose length is given by the second argument, padding to the right with blanks.

8.4.15 **$zseek(arg)**

Function takes one argument (a positive integer) which is a byte offset in the currently active (use) file. The command moves the file pointer to that location in the file. **$zseek()** may only be used on files opened with **old** attribute. Figure 19 gives examples.

```
1  #!/usr/bin/mumps
2   open 1:"tdb,new"
3   for j=1:1:1000 do
4   . use 1
```

```
5   . set i=$ztell
6   . set ^a(j)=i
7   . write "**** ",j,!
8
9   close 1
10  open 1:"tdb,old"
11  for j="":$order(^a(j)):"" do
12  . use 1
13  . set i=$zseek(^a(j))
14  . read a
15  . use 5
16  . write a,!

output:

**** 1
**** 10
**** 100
**** 1000
**** 101
**** 102
**** 103
**** 104
**** 105
**** 106
**** 107
**** 108
**** 109
**** 11
**** 110
**** 111
...
```

Figure 19 $Zseek() Examples

8.4.16 $zsrand(arg)

Seed the random number generator. The value passed as the argument will seed the internal random number generator. If the random number generator is re-seeded with the same seed, the sequence of random numbers produced by **$random** will be the same. The value passed must be a positive integer.

8.4.17 $zstem(arg)

Returns an word English word stem of the argument. This function attempts to remove common endings from words and return a root stem.

8.4.18 $zsystem(arg)

Executes "arg" in a system shell. Returns -1 (fork failed) or the return code of the execution of the argument. See also the **shell** command.

8.4.19 $ztell

Function returns the byte offset in the currently open file. Similar to the C++ **ftello** function. Note: The offset returned is for the file most recently made the default i/o file by the **use** command. **$ztell** may be used on either a file opened as **new, old** or **append**. (See example under **$zseek** above)

8.4.20 $zu(expression)

Function returns 1 if the expression is numeric, 0 otherwise.

8.4.21 $zwi(arg)

Function loads an internal buffer with the string given as the argument. The alphabetic characters of the argument are converted to lower case. The contents of this buffer are returned by the **$zwn** and **$zwp** functions. Figure 20 gives examples.

8.4.22 **$zwn** extract words from buffer

Function returns successive words from the internal buffer delimited by blanks. When no more words remain, it returns an empty string (string of length zero). Returned words are converted to lower case. See **$zwi**.

8.4.23 $zwp extract words from buffer

Function returns successive words from an internal buffer delimited by blanks and punctuation characters. When no more words remain, it returns an empty string (string of length 0). Returned words are converted to lower case. See **$zwi**.

8.4.24 $zws(string) initialize internal buffer

Initializes the parse buffer but does not convert "string" to lower case as is the case with **$zwi**

```
1 #!/usr/bin/mumps
2  set i="now, is the time, for all good"
3  set %=$zwi(i)
4  for w=$zwp write w,!
5  write "-------",!
6  set %=$zwi(i)
7  for w=$zwn write w,!

output:

     now
     ,
     is
     the
     time
     ,
     for
     all
     good
     -------
     now,
     is
     the
     time,
     for
     all
     good
```

Figure 20 $Zwi() Examples

8.4.25 Scan Functions

8.4.25.1.1 $zzScan

8.4.25.1.2 $zzScanAlnum

8.4.25.1.3 $zzInput(var)

The functions return the next word in the current input stream delimited by white space. Words are restricted to a maximum length of 1023. Successive calls return successive words. When there are no more input words, an empty string is returned and **$test** is set to *false*.

If only part of a line is scanned as a result of these functions, a subsequent **read** command will begin at the white space following the last word returned.

If scanning input from stdin (i/o unit 5), you may signal end of file with a *control-d* on a separate line by itself. This will result terminate the scan and **$test** will be set to false.

$zzScan returns all words delimited by whitespace with no conversion. Words may contain any *printable* ASCII character.

$zzScanAlnum processes words before returning them according to the following rules:

- Special characters at the beginning of a word are ignored.
- Words beginning with digits are not returned. If a word begins with one or more special characters followed by a digit, it is not returned.
- Words shorter than 3 characters or longer than 25 characters are not returned.
- Words are converted to all lower case characters.
- If a word contains embedded special characters, it is treated as a delimiter.

Both functions will advance to additional lines as needed. If a word exceeds 1023 bytes, the results are undefined. See Figure 21 for an example.

```
for the input line:

now -- __ ?? !@#$%^&*()_+= IS 2for the time for


      for  set i=$zzScan quit:'$test  write i,!

output:

      now
      --
      __
      ??
      !@#$%^&*()_+=
      IS
      2for
      the
      time
      for

      for  set i=$zzScanAlnum quit:'$test  write i,!

output:

      now
      the
      time
      for


      for i=$zzScanAlnum do
       . write i,!

output:

      now
      the
      time
```

Figure 21 Scan Functions Examples

$zzInput(var) reads an entire input line, converts all characters to lower case, separates the words, removes punctuation (as defined by the C *ispunct()* function except hyphen), and stores the words into a numerically indexed array whose name is the value of the variable or constant passed as the argument. The function returns the number of elements in the array. A return of zero indicates no input was obtained (end of file). As the array created by the function could be quite large, you should probably **kill** it when no it is longer needed. The maximum line length permitted is twice the system parameter *MAX_STR* (9,000 bytes by default).

8.5 Vector and Matrix Functions

8.5.1 $zzAvg(vector)

Computes and returns the average of the numeric values in the vector. For example, see Figure 22.

```
1 #!/usr/bin/mumps
2  for i=1:1:10 set ^a(99,i)=i
3  set i=$zzAvg(^a(99))
4  write "average=",i,!
```

Figure 22 $zzAvg() Example

The above writes 5.5

8.5.2 $zzCentroid(gblMatrix,gblRef)

A centroid vector *gblRef* is calculated for the invoking two dimensional global array *gblMatrix*. The centroid vector is the average value for each for each column of the matrix. Any previous contents of the global array named to receive the centroid vector are lost. The global array *gblMatrix* must contain at least two dimensions. See Figure 23 for an example. The matrix must be a top level global array.

```
1 #!/usr/bin/mumps
2  for i=0:1:10 do
3  . for j=1:1:10 do
4  .. set ^A(i,j)=5
5  set %=$zzCentroid(^A,^B)
6  for i=1:1:10 write ^B(i),!

output:

5
5
5
5
5
5
5
5
5
5
```

Figure 23 $zzCentroid() Example

8.5.3 $zzCount(gblVector)

Counts the number of nodes that contain a value in the global array reference and any descendants. For example, see Figure 17.

```
1 #!/usr/bin/mumps
2  kill ^a
3  for i=1:1:10 set ^a(99,i)=i
4  set i=$zzCount(^a(99))
5  write "count=",i,!
```

```
writes: count=10
```

Figure 24 $zzCount() Example

8.5.4 $zzMax(gbl)

Computes and returns the maximum numeric value in the vector and any descendants. See Figure 25 for an example.

```
1 #!/usr/bin/mumps
1  for i=1:1:10 set ^a(99,i)=i
2  set i=$zzMax(^a(99))
3  write "max=",i,!

output:

10
```

Figure 25 $zzMax() Example

The above writes the largest value stored in the vector.

8.5.5 $zzMin(gbl)

Returns the minimum numeric value stored in the vector and any descendants. See Figure 26 for an example.

```
1 #/usr/bin/mumps
2  for i=1:1:10 set ^a(99,i)=i*2
3  set i=$zzMin(^a(99))
4  write "min=",i,!

output:

2
```

Figure 26 $zzMin() Example

8.5.6 $zzMultiply(gbl1,gbl2,gbl3)

Multiplies the first and second matrix leaving the result in the third. The ordinary rules of algebra apply. Figure 30 gives an example. The arguments *gbl1* and *gbl2* must be top level, two dimensional arrays.

8.5.7 $zzSum(gblVector)

Computes and returns the sum of the numeric values stored in the vector. For example, see Figure 31.

8.5.8 $zzTranspose(gblMatrix1,gblMatrix2)

Transposes the first global array matrix leaving the result in the second. For example, see Figure 32. the argument *gblMatrix1* must be a top level, two dimensional array.

8.6 Text Processing Functions

The following functions are used in connection with experiments in information storage and retrieval.

8.6.1 Similarity Functions

8.6.1.1 $zzCosine(gbl1,gbl2)

8.6.1.2 $zzSim1(gbl1,gbl2)

8.6.1.3 $zzDice(gbl1,gbl2)

8.6.1.4 $zzJaccard(gbl1,gbl2)

These compute the Cosine, Sim1, Dice and Jaccard similarity coefficients between document vectors given as the first and second arguments. Both arguments are numeric global array vectors. The formulae are given in Figure 27 and an example in code is given in Figure 28. The formulae calculate the similarities between two global array vector *gbl1* and global array vector *gbl2*. The vectors need not be of equal length. Missing elements are interpreted as zero. The vectors should be top level vectors.

$$Simularity_{Dice}(i,j)=\frac{2\sum_{k=1}^{k=t} Term_{ik}\cdot Term_{jk}}{\sum_{k=1}^{k=t} Term_{ik}+\sum Term_{jk}}$$

$$Simularity_{Jaccard}(i,j)=\frac{\sum_{k=1}^{k=t} Term_{ik}\cdot Term_{jk}}{\sum_{k=1}^{k=t} Term_{ik}+\sum Term_{jk}-\sum_{k=1}^{k=t}(Term_{ik}\cdot Term_{jk})}$$

$$Simularity_{Cosine}(i,j)=\frac{\sum_{k=1}^{k=t} Term_{ik}\cdot Term_{jk}}{\sqrt{\sum_{k=1}^{k=t} Term_{ik}^2\cdot\sum Term_{jk}^2}}$$

$$Simularity_{¿1}(i,j)=\sum_{k=1}^{k=t} Term_{ik}\cdot Term_{jk}$$

Figure 27 Similarity Formulae

```
1  #!/usr/bin/mumps
2   kill ^A
3   kill ^B
4
5   set ^A("1")=3
6   set ^A("2")=2
7   set ^A("3")=1
8   set ^A("4")=0
9   set ^A("5")=0
10  set ^A("6")=0
11  set ^A("7")=1
12  set ^A("8")=1
13
14  set ^B("1")=1
15  set ^B("2")=1
16  set ^B("3")=1
17  set ^B("4")=0
```

```
18  set ^B("5")=0
19  set ^B("6")=1
20  set ^B("7")=0
21  set ^B("8")=0
22
23  write "Cosine=",$zzCosine(^A,^B),!
24  write "Sim1=",$zzSim1(^A,^B),!
25  write "Dice=",$zzDice(^A,^B),!
26  write "Jaccard=",$zzJaccard(^A,^B),!

output:

Cosine=0.75
Sim1=6
Dice=1
Jaccard=1
```

Figure 28 Similarity Functions

8.6.2 $zzBMGSearch(arg1,arg2)

Boyer-Moore-Gosper Function returns the number of non-overlapping occurrences of *arg1* in *arg2*.

These functions, were obtained from

```
ftp://ftp.uu.net/usenet/comp.sources.unix/volume5/bmgsubs.Z
```

and were written by Jeffrey Mogul (Stanford University), based on code written by James A. Woods (NASA Ames, an agency of the U.S. Government) and are thus believed to be in the public domain. Figure 29 gives an example.

```
1 #!/usr/bin/mumps
2  set key=”now”
3  set str=”now is the now of the now in the know”
4  write $zBMGSearch(key,str),!

output:

4
```

Figure 29 $zzBMGSearch() Example

8.6.3 $zPerlMatch(string,pattern)

Applies the Perl **pattern** to **string** and returns 1 if the pattern fits and 0 otherwise. The **$zPerlMatch** function has the side effect of creating variables in the local symbol table to hold backreferences, the equivalent concept of **$1**, **$2**, **$3**, ... in Perl. Up to nine backreferences are currently supported, and can be accessed through the same naming scheme as Perl (**$1** through **$9**). These variables remain defined up to a subsequent call to **$zPerlMatch**, at which point they are replaced by the backreferences captured from that invocation. Undefined backreferences are cleared between invocations; that is, if a match operation captured five backreferences, then $6 through $9 will contain the empty string. Figure 33 contains examples (long lines wrapped).

```
1  #/usr/bin/mumps
2   set ^d("1","1")=2
3   set ^d("1","2")=3
4   set ^d("2","1")=1
5   set ^d("2","2")=-1
6   set ^d("3","1")=0
7   set ^d("3","2")=4
8
9   set ^e("1","1")=5
```

```
10  set ^e("1","2")=-2
11  set ^e("1","3")=4
12  set ^e("1","4")=7
13  set ^e("2","1")=-6
14  set ^e("2","2")=1
15  set ^e("2","3")=-3
16  set ^e("2","4")=0
17
18  set %=$zzMultiply(^d,^e,^f)
19
20  for i="":$order(^f(i)):"" do
21  . for j="":$order(^f(i,j)):"" do
22  .. write i," ",j," ",^f(i,j),!
```

output:

```
1 1 -8
1 2 -1
1 3 -1
1 4 14
2 1 11
2 2 -3
2 3 7
2 4 7
3 1 -24
3 2 4
3 3 -12
3 4 0
```

Figure 30 $zzMultiply() Example

```
1 #!/usr/bin/mumps
2  for i=1:1:10 set ^a(99,i)=i
3  set i=$zzSum(^a(99))
4  write "sum=",i,!
```

output:

```
55
```

Figure 31 $zzSum() Example

```
1  #!/usr/bin/mumps
2   kill ^f
3
4   set ^d("1","1")=2
5   set ^d("1","2")=3
6   set ^d("2","1")=4
7   set ^d("2","2")=0
8
9   set %=$zzTranspose(^d,^f)
10
11  for i="":$order(^f(i)):"" do
12  . for j="":$order(^f(i,j)):"" do
13  .. write i," ",j," ",^f(i,j),!
```

output:

```
     1 1 2
     1 2 4
     2 1 3
     2 2 0
```

Figure 32 $zzTranspose() Example

```
1 #!/usr/bin/mumps
2  write "Please enter a telephone number:",!
3   read phonenum
4
5   set p="^(1-)?(\(?\d{3}\)?)?(-| )?\d{3}-?\d{4}$"
6   if $zperlmatch(phonenum,p) do
7   . write "+++ This looks like a phone number.",!
8   . write "The area code is: ",$2,!
9   else  do
10  . write "--- This didn't look like a phone number.",!

output:

Please enter a telephone number:
(123) 456-7890
+++ This looks like a phone number.
The area code is: (123)

Please enter a telephone number:
(123) 456-7890
+++ This looks like a phone number.
```

Figure 33 $zPerlMatch() Example

8.6.4 $zReplace(string,pattern,replacement)

The regular expression in *pattern* is evaluated on *string* and, if there is a match, the matching section is replaced by *replacement*. Figure 34 contains an example. In the first part, the word 'is' is replaced by 'IS'. In the second part, a match is sought for any content between two sets of matching brackets ([[...]]). The matched section is in back reference **$2**. This is then used as a pattern to be replaced.

8.6.5 $zShred(string,length)

8.6.6 $zShredQuery(string,length)

The **$zShred()** function segments the input argument **string** into fragments of **length** size upon successive calls. The function returns a string of length zero when there are no more fragments of size **length** remaining (thus, short fragments at the end of a string are not returned).

$zShred copies the input string to an internal buffer upon the first call. Subsequent calls retrieve from this buffer. When the buffer is consumed, the function will copy the contents of the next string submitted to the buffer. Figure 35 contains an example.

```
1 #!/usr/bin/mumps
2  set a="now is the time for all"
3  set a=$zReplace(a,"is","IS")
4  write a,!
5
6  set a="[[now is the time]]"
7  if $zPerlMatch(a,"(\[\[)(.*)(\]\])") do
8  . set a=$zReplace(a,$2,"ABC")
9  . write a,!

output:

now IS the time for all
[[ABC]]
```

Figure 34 $zReplace() Example

```
1 #!/usr/bin/mumps
2  set a="now is the time for all good men to "
3  set a=a_"come to the aid of the party"
4  for  do  quit:j=""
5  . set j=$zShred(a,5)
6  . if j="" quit
7  . write j,!

output:

     nowis
     theti
     mefor
     allgo
     odmen
     tocom
     etoth
     eaido
     fthep
```

Figure 35 $zShred() Example

The **$zShredQuery** function segments **length** shifted copies of the input **string** into fragments of size **length** upon successive calls. That is, the function first returns all the fragments of size length of the **string** in the same manner as **$zShred**. However, it then shifts the starting point of the input string to the right by one and returns all the fragments of size length relative to the shifted starting point. If repeatedly called, it repeats this process a total of **length** times. When there are no more combinations, the empty string is returned as shown in Figure 36.

```
1 #!/usr/bin/mumps
2  set a="now is the time for all good men to come to "
3  set a=a_"the aid of the party"
4  for  do  quit:j=""
5  . set j=$zShredQuery(a,5)
6  . if j="" quit
7  . write j,!

output:

     nowis                     tothe                     goodm
     theti                     aidof                     entoc
     mefor                     thepa                     ometo
     allgo                     wisth                     theai
     odmen                     etime                     dofth
     tocom                     foral                     epart
     etoth                     lgood                     isthe
     eaido                     mento                     timef
     fthep                     comet                     orall
     owist                     othea                     goodm
     hetim                     idoft                     entoc
     efora                     hepar                     ometo
     llgoo                     isthe                     theai
     dment                     timef                     dofth
     ocome                     orall                     epart
```

Figure 36 $ShredQuery() Example

8.6.7 $zzSoundex(s1)

Returns the Soundex code for the argument string as follows:

1. All letters are converted to lower case;
2. Non-alphabetic characters are removed;
3. Adjacent duplicate letters are replaced by a single occurrence;
4. The first letter is retained;
5. The letters b, f, p, and v are replaced by the number 1;
6. The letters c, g, j, k, q, s, x, and z are replaced by the number 2;
7. The letters d and t are replaced by the number 3;
8. The letter l is replaced by the number 4;
9. The letters m and n are replaced by the letter 5;
10. the letter r is replaced by the number 6;
11. The is truncated to four characters.

8.6.8 $zSmithWaterman(s1,s2,algn,mat,gap,noMatch,match)

Computes the Smith Waterman score between two strings. Result returned is the highest alignment score achieved. String lengths are limited by **STR_MAX** in the interpreter. If you compare very long strings (>100,000 characters), you may exceed stack space. This can be increased under Linux with the command:

```
ulimit -s unlimited
```

Figure 37 gives an example.

```
1 #!/usr/bin/mumps
2  set s1="now is the time"
3  set s2="now  i th  time"
4  set i=$zSmithWaterman(s1,s2,1,0,-1,-1,2)
5  write "score=",i,!

output:

   1 now- is the time 16
     ::: :: ::: :::::
   1 now  i- th  time 16

score=23
```

Figure 37 $zSmithWaterman() Example

Parameters:

If *algn* is zero, no printout of alignments is produced. If *algn* is not zero, a summary of the alternative alignments will be printed.

If *mat* is zero, intermediate matrices will not be printed.

The parameters *gap, noMatch* and *match* are the gap and mismatch penalties (negative integers) and the match reward (a positive integer).

If insufficient memory is available, a segmentation violation will be raised. Try increasing your stack size.

8.6.9 $zzIDF(global,doccount)

Calculates the Inverse Document Frequency score of words contained in the argument *global*. The parameter *doccount* is the total number of documents. The index of each element of the *global* vector is a word and the value stored is the number of times the word occurs in the collection. Figure 38 gives and example. The vector argument *global* must be a top level array.

```
1 #!/usr/bin/mumps
2  set ^a("now")=2
3  set ^a("is")=5
4  set ^a("the")=6
```

```
5  set ^a("time")=3
6  set j=4
7  set %=$zzIDF(^a,j)
8  for i="":$order(^a(i)):"" write i," ",^a(i),!

output:

is 0.7
now 2.0
the 0.4
time 1.4
```

Figure 38 $zzIDF() Example

8.6.10 Correlation Functions

8.6.10.1 $zzTermCorrelate(global1,global2)

Calculates the Term-Term co-occurrence matrix for the Document-Term matrix in *global1*. The result is placed in *global2*.

A Term-Term matrix has terms (words) as the indices of its rows and columns. A Term-Term matrix gives, for each position, the degree to which the term corresponding to the row is similar to the term corresponding to the column. The diagonal, which is the degree a term is related to itself, is ignored. Both operands must be top level arrays.

In both the doc-doc and term-term matrices, the upper and lower diagonal matrices are mirror images of one another. Figure 39 gives an example. The order of words in the output will depend upon which data base facility is in use and what it's collating settings are. The Native global array handler collates according to ASCII-7.

```
1  #!/usr/bin/mumps
2   kill ^A,^B
3
4   set ^A("1","computer")=5
5   set ^A("1","data")=2
6   set ^A("1","program")=6
7   set ^A("1","disk")=3
8   set ^A("1","laptop")=7
9   set ^A("1","monitor")=1
10
11  set ^A("2","computer")=5
12  set ^A("2","printer")=2
13  set ^A("2","program")=6
14  set ^A("2","memory")=3
15  set ^A("2","laptop")=7
16  set ^A("2","language")=1
17
18  set ^A("3","computer")=5
19  set ^A("3","printer")=2
20  set ^A("3","disk")=6
21  set ^A("3","memory")=3
22  set ^A("3","laptop")=7
23  set ^A("3","USB")=1
24
25  set %=$zzTermCorrelate(^A,^B)
26
27  for i="":$order(^B(i)):"" do
28  . write i,!
29  . for j="":$order(^B(i,j)):"" do
30  .. write ?10,j," ",^B(i,j),!
```

output:

```
USB                                     monitor 1               monitor
        computer 1                      printer 1                       computer 1
        disk 1                          program 1   language            data 1
        laptop 1                        computer 1                      disk 1
        memory 1                        laptop 1                        laptop 1
        printer 1                       memory 1                        program 1
computer                                printer 1               printer
        USB 1                           program 1                       USB 1
        data 1                  laptop                                  computer 2
        disk 2                          USB 1                           disk 1
        language 1                      computer 3                      language 1
        laptop 3                        data 1                          laptop 2
        memory 2                        disk 2                          memory 2
        monitor 1                       language 1                      program 1
        printer 2                       memory 2                program
        program 2                       monitor 1                       computer 2
data                                    printer 2                       data 1
        computer 1                      program 2                       disk 1
        disk 1                  memory                                  language 1
        laptop 1                        USB 1                           laptop 2
        monitor 1                       computer 2                      memory 1
        program 1                       disk 1                          monitor 1
disk                                    language 1                      printer 1
        USB 1                           laptop 2
        computer 2                      printer 2
        data 1                          program 1
        laptop 2
        memory 1
```

Figure 39 $zTermCorrelate() Example

8.6.10.2 $zzDocCorrelate(gblref1,gblref2,mthd,thrshld)

A square Document-Document matrix *gblref2* is calculated from the Document-Term matrix *gblref1* according to method *mthd* (Cosine, Sim1, Dice, Jaccard). The value of elements in the Document-Document matrix will not exceed threshold (*thrshld*) and the cells associated with coresponding document numbers will not exist.

A Document-Document matrix has document id's as its row and column indices. A cell in the matrix indicates the degree to which the row document is related to the column document. The diagonal is ignored. Figure 40 gives an example.

8.6.11 Stop and Synonym Functions

8.6.11.1 $zStopInit(arg)

8.6.11.2 $zStopLookup(word)

8.6.11.3 $zSynInit(fileName)

8.6.11.4 $zSynLookup(word)

A call to **$zStopInit(file_name)** will open and load a file of stop words into a C++ container. The file should consist of one word per line. If the file cannot be opened or there is insufficient memory to hold the list of words, the program will halt with an error message. **$zStopInit()** converts all words to lower case.

```
1  #!/usr/bin/mumps
2    kill ^A,^B
3
4    set ^A("1","computer")=5
5    set ^A("1","data")=2
6    set ^A("1","program")=6
7    set ^A("1","disk")=3
8    set ^A("1","laptop")=7
```

```
9   set ^A("1","monitor")=1
10
11  set ^A("2","computer")=5
12  set ^A("2","printer")=2
13  set ^A("2","program")=6
14  set ^A("2","memory")=3
15  set ^A("2","laptop")=7
16  set ^A("2","language")=1
17
18  set ^A("3","computer")=5
19  set ^A("3","printer")=2
20  set ^A("3","disk")=6
21  set ^A("3","memory")=3
22  set ^A("3","laptop")=7
23  set ^A("3","USB")=1
24
25  set %=$zzDocCorrelate(^A,^B,"Cosine",.5)
26
27  for i="":$order(^B(i)):"" do
28  . write i,!
29  . for j="":$order(^B(i,j)):"" do
30  .. write ?10,j," ",^B(i,j),!

output:

1
         2 0.887096774193548
         3 0.741935483870968
2
         1 0.887096774193548
         3 0.701612903225806
3
         1 0.741935483870968
         2 0.701612903225806
```

Figure 40 $zDocCorrelate()Example

A call to **$zStopLookup(word)** will return 1 if *word* is in the stop list, 0 otherwise. Words presented to **$zStopLookup(word)** should be in lower case.

$SynInit() opens a synonym file. The file should consist of two or more words per line separated by from one another by one blank. The words are treated as synonyms with the first word on each line as the primary synonym. The primary synonym may be a code or category number. This word or code will be returned if any of the remaining words are passed as arguments to **$SynLookup()**. Figure 41 gives an example.

8.7 SQL functions

These functions are peculiar to this implementation.

8.7.1 $zsql

Returns the SQL server error message for the most recent command or 'ok.'

8.7.2 $zsqlCols

Returns a string consisting of the columns names for the most recent operation that returned tuples. Each name is separated from the next by a TAB character ($char(9)).

```
Assume that the file “stop” contains the word “and”

 set %=$zStopInit("stop")
```

```
 if $zStopLookup("and") write "yes",!

Writes yes

Assume that the file “synonyms” contains a line with the text:

compression compressions compress compressed compresses

 set %=$zSynInit("synonyms")
 write $zSynLookup("compressions"),!

output:

compression
```

Figure 41 Stop List Functions

8.7.3 $zsqlOpen

Returns *true* if a connection to the SQL server is open, *false* otherwise.

8.7.4 $zNative

$znative returns true if globals are being stored in the native global array.

8.7.5 $zMysql

$zmysql returns true if globals are being stored in MySQL

8.7.6 $zPostgres

$zpostgres returns true if globals are being stored in PostgreSQL

8.7.6.1 $zTable

$ztable returns a comma separated string. The portion prior to the comma is the current RDBMS table in which the Mumps globals are stored. The part after the comma is the maximum number of indices permitted in the table (same as *$ztabsize*).

$ztable may be set. If it is set immediately prior to an *sql/f* command, it is the name of the table to be created and/or initialized which now becomes the default table.

If *$ztable* is set to the name of a table which exists, global array reference will be made to this table until *$ztable* is changed. When *$ztable* is changed to the name of an existing table, *$ztabsize* is updated to the value for the now current table.

8.7.7 $zTabsize

$ztabsize returns the number of RDBMS columns available for global array indexes. May be set immediately prior to an *sql/f* command in which case the value in *$ztabsize* will be used to set the number of columns (range: 1 to 20).

8.7.8 $zsqlOutput

$zsqlOutput contains the name of the file to which output from SQL commands will be written. It may assigned a string value that will be used as the SQL command processor output file by the next SQL command.

9 Pattern Matching

9.1 Mumps 95 Pattern Matching

Author: Matthew Lockner

Mumps 95 compliant pattern matching (the '?' operator) is implemented in this compiler/interpreter as given by the following grammar:

```
pattern         ::= {pattern_atom}
pattern_atom    ::= count pattern_element
count           ::= int | '.' | '.' int | int '.' | int '.' int
pattern_element ::= pattern_code {pattern_code} | string | alternation
pattern_code    ::= 'A' | 'C' | 'E' | 'L' | 'N' | 'P' | 'U'
alternation     ::= '(' pattern_atom {',' pattern_atom} ')'
```

The largest difference between the current and previous standard is the introduction of the alternation construct, an extension that works as in other popular regular expressions implementations. It allows for one of many possible pattern fragments to match a given portion of subject text.

A string literal must be quoted. Also note that alternations are only allowed to contain pattern atoms and not full patterns; while this is a possible shortcoming, it is in accordance with the standard. It is a trivial matter to extend alternations to the ability to contain full patterns, and this may be implemented upon sufficient demand.

Pattern matching is supported by the Perl-Compatible Regular Expressions library (PCRE). Mumps patterns are translated via a recursive-descent parser in the Mumps library into a form consistent with Perl regular expressions, where PCRE then does the actual work of matching. Internally, much of this translation is simple character-level transliteration (substituting '|' for the comma in alternation lists, for example). Pattern code sequences are supported using the POSIX character classes supported in PCRE and are mostly intuitive, with the possible exception of 'E', which is substituted with `[[:print][:cntrl:]]`. Currently, this construct should cover the ASCII 7-bit character set (lower ASCII).

Due to the heavy string-handling requirements of the pattern translation process, this module uses a separate set of string-handling functions built on top of the C standard string functions, using no dynamic memory allocation and fixed-length buffers for all operations whose length is given by the constant STR_MAX in *sysparms.h*. If an operation overflows during the execution of a Mumps compiled binary, a diagnostic is output to *stderr* and the program terminates. If such termination occurs too frequently, simply increase the value of STR_MAX.

9.2 Using Perl Regular Expressions

Author: Matthew Lockner

In addition to Mumps 95 pattern matching using the '?' operator, it is also possible to perform pattern matching against Perl regular expressions via the *perlmatch* function. Support for this functionality is provided by the Perl-Compatible Regular Expressions library (PCRE), which supports a majority of the functionality found in Perl's regular expression engine.

The *perlmatch* function works in a somewhat similar fashion to the '?' operator. It is provided with a subject string and a Perl pattern against which to match the subject. The result of the function is boolean and may be used in boolean expression contexts such as the "If" statement.

Some subtleties that differ significantly from Mumps pattern matching should be noted:

1. A Mumps match expects that the pattern will match against the entire subject string, in that successful matching implies that no characters are left unmatched even if the pattern matched against an initial segment of the subject string. Using *perlmatch*, it is sufficient that the entire Perl pattern matches an initial segment of the subject string to return a successful match.

2. The *perlmatch* function has the side effect of creating variables in the local symbol table to hold *backreferences*, the equivalent concept of $1, $2, $3, ... in Perl. Up to nine backreferences are currently supported, and can be accessed through the same naming

scheme as Perl ($1 through $9). These variables remain defined up to a subsequent call to *perlmatch* , at which point they are replaced by the backreferences captured from that invocation. Undefined backreferences are cleared between invocations; that is, if a match operation captured five backreferences, then $6 through $9 will contain the null string.

Examples

This program asks the user to input a telephone number. If the data entered looks like a valid telephone number, it extracts and prints the area code portion using a backreference; otherwise, it prints a failure message and exits.

```
Write "Please enter a telephone number:",!
Read phonenum

If $$^perlmatch(phonenum,"^(1-)?(\(?\d{3}\)?)?(-| )?\d{3}-?\d{4}$") Do
. Write "+++ This looks like a phone number.",!
. Write "The area code is: ",$2,!
Else  Do
. Write "--- This didn't look like a phone number.",!
```

The output of several sample runs of the program follows:

```
Please enter a telephone number:
1-123-555-4567
+++ This looks like a phone number.
The area code is: 123


Please enter a telephone number:
(123)-555-1234
+++ This looks like a phone number.
The area code is: (123)


Please enter a telephone number:
(123) 555-0987
+++ This looks like a phone number.
The area code is: (123)
```

As in Perl, sections of the regular expression contained in parentheses define what is contained in the backreferences following a match operation. The backreference variables are named in a left-to-right order with respect to the expression, meaning that $1 is assigned the portion matched against the leftmost parenthesized section of the regular expression, with further references assigned names in increasing order. For a much more in-depth treatment of the subject of Perl regular expressions, refer to the *perlre* manpage distributed with the Perl language (also widely available online).

10 Mumps Compiler

NOTE: This section is being re-written and the information currently in it may be unreliable.

Included in the distribution package is a beta version compiler for the Mumps language. At present, not all Mumps language features are implemented but many are.

The compiler translates Mumps to to C++ and then compiles the C++ programs and the result is an executable binary version of the program.

10.1 Compiling Programs

The Mumps programs described in this document can be run in either of two ways: either as interpreted code using the Mumps interpreter or as binary executables resulting from the Mumps Compiler.

Binary programs run faster than interpreted programs but the difference can be small if the programs rely heavily on input/output operations.

10.2 How to Compile and Run a Program.

Programs written in Mumps must have the extension *.mps* when used with the compiler. Programs written for the interpreter, however, may have any extension but *.mps* is preferred. MDH programs written in C++ must have the ".cpp" extension.

When you compile a Mumps program, a C++ translation of your program is created and resides on the disk with the same name but with the *.cpp* extension. The C++ translation is then compiled and linked with run-time libraries to build an executable binary.

On MS Windows, the binary will have the same name as your original program but with the .exe extension. On Linux, the binary will have the same name as your original program but with no extension. Depending on which system you are using, there will be other, intermediate files generated by the Mumps and C++ compilers. These are not important and can be deleted.

You may compile a program either by using the built in script *mumpsc*.

To compile a Mumps program using the script, type:

```
mumpsc myprog.mps
```

This will translate your Mumps program to C++, run the C++ compiler on the result, and link the output of the C++ compiler with the standard Mumps libraries.

This script file runs the actual compiler, *mumps2c* which translates the Mumps program to C++ whose poutput will be named *myprog.cpp*.

Subsequently, a second command runs the C++ compiler on *myprog.cpp* and links the result with the standard Mumps libraries.

Also, use the above "g++" command to compile a C++ program generated by the Mumps/II Compiler that you may have edited (for example, to insert debugging information). You may not use the "mumpsc" script to compile a ".cpp" program generated by the Mumps/II Compiler. The "mumpsc" script may only be used to compile Mumps source programs (".mps" extensions) and MDH programs (".cpp" extensions). When "mumpsc" sees you compiling a program with the ".cpp" extension, it sets certain switches that are not appropriate for a ".cpp" program generated by the Mumps Compiler.

To compile an MDH/C++ program, type:

```
mumpsc myprog.cpp
```

To interpret a program, type:

```
mumps myprog.mps
```

Generally speaking, in most cases you will receive syntax error messages from the Mumps compiler which will identify the error and the line number in the original Mumps program containing the error.

However, when using the compiler, in some cases, an error may be detected by the C++ compiler. If you get C++ error messages, the line number on the error message will refer to the line number in the C++ translation of your Mumps program. To translate this to a line number in your Mumps program, look into the generated *.cpp* file at the line number given in the C++ error message and then back track to the nearest prior commented Mumps source line - this is the line in your Mumps programs that caused the problem.

For example, if you get a message from the C++ compiler saying that you have an error at line 1234 in the C++ module, open the C++ file and move to line 1234. At that location you may see something like:

```
/*================================================================================*
svPtr->LineNumber=4; //       write "the sum is: ",total,!
/*================================================================================*/
       if (svPtr->out_file[svPtr->io]==NULL) ErrorMessage("Write to input file",svPtr->LineNumber);
       svPtr->hor[svPtr->io]+=fprintf(svPtr->out_file[svPtr->io],"%s","the sum is: ");
       if (sym_(SYMGET,(unsigned char *) "total",(unsigned char *) tmp0,svPtr)==NULL)
            VariableNotFound(svPtr->LineNumber);
       svPtr->hor[svPtr->io]+=fprintf(svPtr->out_file[svPtr->io],"%s",tmp0);
       fprintf(svPtr->out_file[svPtr->io],"\n"); svPtr->hor[svPtr->io]=0; svPtr->ver[svPtr->io]++;
```

Figure 42 Example C++ Code

Notice that each original line of Mumps code and its line number from the original Mumps file appear in a comment prior to the C++ translation of the line.

Thus, to locate the line of Mumps code that caused the C++ error, look for the line of Mumps code preceding the line which the C++ complier flagged as being in error.

Generally speaking, you may receive C++ error messages if you reference non-existent labels or subroutines, or incorrectly specify indented do blocks (see below).

Also, you may see ^M (control-M) characters in the code, especially if you are viewing a MS WinXP file with a Linux editor. These are visible due the differences between the operating systems. Under WinXP, each line ends in a *carriage-return* and a *line-feed*. Under Linux, each line ends in a *line-feed* character only. The control-M's you see are the carriage-returns. They are harmless.

10.3 Global Array Storage in Compiled Programs

Global arrays will be stored in PostgreSQL, MySQL or the native Btree database depending on which script you used to build the interpreter (see 1.9 on page 9). Global arrays created by compiled programs are interchangeable with global arrays created by the interpreter.

10.4 Compiler Implementation Overview

The compiled modules execute between 1.5 and 6 times faster than the same code executing on an interpreter. The lower multiplier reflects benchmark programs which are very global array intensive while the higher multiple is for programs that are less global array intensive.

One advantage of full compilation is interoperability with other languages and with the host operating system. Programs written in C++ have full access to all system features.

10.4.1 Mumps Main Programs and Functions

When you compile your Mumps program, one or more C++ functions will be created. These, in turn, are compiled into binary executable code.

A compiled Mumps program consists of a main routine along with multiple functions that are called upon to provide services.

You may write your own functions. There are three ways to include your functions into a Mumps program:

1. As Mumps internal functions,

2. As Mumps external functions, or
3. As C++ functions.

11 Multi-Dimensional and Hierarchical Database Class Library (MDH)

The Multi-Dimensional and Hierarchical Database Toolkit (MDH) is a Linux-based, open sourced, toolkit of portable libraries that support access to the Mumps multi-dimensional and hierarchical database and other services. The package is written in C and C++ and licensed under the GNU GPL/LGPL licenses.

The toolkit permits manipulation of very large, character string indexed, multi-dimensional, sparse matrices from C++ programs. The toolkit supports access to PostgreSQL and MySQL relational data base servers, the Perl Compatible Regular Expression Library, and the Glade GUI builder.

The toolkit makes Mumps data base and functions available as C++ classes and permits execution of Mumps scripts directly from C++ programs. The toolkit is provided with the Mumps distribution and is available if Mumps is installed. No further installation beyond the basic Mumps installation described above is required.

The class, function and macro libraries primarily operate on global arrays. Global arrays are undimensioned, string indexed, disk resident data structures whose size is limited only by available disk space. They can be viewed either as multi-dimensional sparse matrices or as tree structured hierarchies.

To compile an MDH/C++ program using the script, type:

```
mumpsc myprog.cpp
```

11.1 MDH Class Library Header File

To use the class libraries, add the following to the beginning of your C++ program:

```
#include <mumpsc/libmpscpp.h>
```

This statement inserts in the necessary header files for you C++ program. In addition to the MDH class libraries, the following standard systems headers will be included as well:

```
#include <iostream>
#include <iomanip>
#include <string>
#include <string.h>
#include <math.h>
#include <stdlib.h>
```

11.2 MDH Data Types

The MDH is built upon two data classes. One is for global arrays (**global**)and the other is a string data type (**mstring**) which mimics that of Mumps strings.

11.2.1 Mstring Data Objects

The **mstring** class provides functionality similar to the basic typeless string data type in Mumps. Objects of **mstring** may contain text, integers and floating point values. Operations on **mstring** objects include addition, multiplication, subtraction, division, modulo, concatenation and so forth. Objects of type **mstring** are declared in the normal manner such as:

```
mstring mvar1,var2,var3;
```

They may be initialized with **int**, **long**, **float**, **double**, **char** * and **string** and **mstring** values such as:

```
mstring var1(10),var2(10.123),var3("test"),var4(stringVar);
```

Objects of type **mstring** may be assigned to most data types and most data types may be assigned to objects of type **mstring**.

Objects of type **mstring**, **string**, and null terminated character strings are the only legal indices for objects of class **global**.

11.2.1.1 Arithmetic Operations on Mstring Objects

When **mstring** objects contain numeric values, you may apply arithmetic operators directly to the **mstring** object or objects.

Both extended precision and basic hardware precision are available.

In hardware precision mode, floating point numbers are processed by the machine's arithmetic processing hardware. Floating point numbers are treated as 64-bit *double* values and integers are treated as signed 64-bit *long* integer values. Thus, integers may range from:

`-9,223,372,036,854,775,808 (-2`^{63}`+1) to 9,223,372,036,854,775,807 (2`^{63}`-1)`

Hardware floating point numbers utilize a one bit sign, an 11 bit exponent and a 52 bit fraction. This translates into approximately 16 decimal digits of precision in the range of ± ~$10^{-323.3}$ to ± ~$10^{308.3}$.

Extended precision is available through use of the GNU multiple precision arithmetic library[22] and the GNU MPFR library[23]. For integers, this means effectively unlimited precision. For floating point, the exponent is 64 bits and the fraction is user specified (default of value of 72 bits).

Hardware arithmetic will be selected during system build if (1) *configure* does not find the extended precision libraries or (2) the user specifies the configuration option:

--with-hardware-math.

If the extended precision libraries are found and the above option has *not* been specified, extended precision will be in effect.

If extended precision is used, the number of bits in the fraction of a floating point number can be set with:

--with-float-bits=value

where *value* is the number of bits. The default value is 72.

For extended precision floating point numbers, the number of digits of precision that may be printed is controlled by:

--with-float-digits=value

where *value* is the number of digits. The default is 20.

When printing an extended precision floating point number, the number of digits being printed should be consistent with the number of bits in the fraction. If the number of digits is too large, insignificant, random low-order digits may appear in the output.

11.3 Global Data Objects

Objects of class **global** provide access to the global array database. The class includes functions to create, delete (kill), and navigate global arrays.

In your C++ program, you must declare each global array that the program will use. Normally, these declarations will appear at the beginning of the program. A global declaration has the form:

```
global program_ref(database_name);
```

Where *program_ref* is the name by which the global array will be referred to in your program and *database_name* is the name of the actual global array in the file system. Both may be the same. The value for *database_name* may be expressed as a pointer to a character string constant.

22 http://www.mpfr.org/
23 http://gmplib.org/manual/index.html

For example, if your program uses a Mumps global array stored in the file system with the name *patient*, you might have the following C++ declaration in your program:

```
global patient("patient");
```

Once declared, a global array object may be used to access the contents of the global array database. For example, for the global array object *patient* declared above, the following reference might be made:

```
patient(ptid,test,date,time)=result;
```

where *ptid, test, data, result* and *time* are **mstring** or **char *** null terminated variables or constants.

Although objects of class **mstring** may be C++ arrays, objects of class **global** may not.

Objects of class **global** may *not* be initialized in declaration statements**.**

11.4 Operators Defined on Mstring & Global Objects

Objects of class **mstring** may appear as the operands of most C++ builtin operators by means of C++ operator overloading.

In the cases of binary operators, the other operand may be most other builtin data types as well as **global** and **mstring** objects.

Figure 43 contains the full list of C++ operators that have been overloaded for use with objects of types **mstring** and **global**. In these examples, assume the declarations:

```
mstring ms, msa[10];
global gb("test");
```

Unary Operators	Description	Examples
++ --	Suffix/postfix increment and decrement	ms++; gb("123")++;
[]	Array subscripting[24]	mstring msa[10]; msa[1] = "abc";
++ --	Prefix increment and decrement	++ms; ++gb("123");
+ -	Unary plus and minus	cout << +gb("123") << endl; cout << -ms << endl;
(*type*)	C-style explicit cast	ms = "123" int k = (int) ms("123");

*	Indirection (dereference)	global *p1 = &gb;
		(*p1)("111") = 10;
		mstring *p2 = msa;
		(*p2)[3] = "abc";

& (unary)	Address-of	mtstring *p1 = &ms;

new, new[]	Dynamic memory allocation	global *p3 = new global("xxx");
		(*p3)("xxx") =2 2;
		mstring *p4 = new mstring;
		*p4=123;

delete, delete[]	Dynamic memory deallocation	delete p1;
Binary Operators[25]	Description	Examples

* / %	Multiplication, division, and remainder	ms = ms * 2;
		ms = gb("123") / ms;
		ms = gb("123") % 5;

24 Only with an **mstring** operand.

25 One operand, the first, may be of type **mstring** or **global** and the other may be of type **mstring**, **global**, **float**, **double**, **int**, **long**, **char***, or **string**.

+ -	Addition and subtraction	ms = ms + 2; ms = gb("123") - ms;
<< >>	stream insertion / extraction	cout << ms; cin >> gb("123");

< <=	For relational operators < and ≤ respectively[26]	if (ms <= gb("123")) ...
		if (ms < gb"abc")) ...
		if ("abc" < gb("123") ...

> >=	For relational operators > and ≥ respectively[26]	if (ms >= gb("123") ...
		if (ms > gb("abc")) ...
		if ("abc" > gb("123")) ...

== !=	For relational operators = and ≠ respectively[26]	if (ms == gb("123") ... if (ms != gb("abc")) ...
&&	Logical AND	if (ms && gb("123")) ...
||	Logical OR	if (ms || gb("123")) ...
Ternary Operator	Description	Examples
?:	Ternary conditional	ms ? ms : y
Assignment[27]	Description	Examples
=	Direct assignment	ms = 123 gb("123") = 1.3456 ms = "test"
+= -=	Compound assignment by sum and difference	ms=0; ms += 123 ms+="123"; gb("123")=0; gb("123") -= 10
*= /= %=	Compound assignment by product, quotient, and remainder	ms=0; ms *= 123 gb("123")=10; gb("123") /= 10 gb("123")=10; gb("123") %= 10
& (binary)	Concatenate. First operand must be of type **global** or **mstring**[28]. The second operand may be **string**, **mstring**, **global**, **char* int**, **long**, or **double**.	mstring i="aaa",j="bbb",k="ccc"; i=i&j&k; // i -> aaabbbccc

Figure 43 Operators Defined on **mstring** and **global**

11.5 Example Arithmetic Operations on global & mstring Objects

The operations of add, subtract, multiply, divide, pre/post increment and pre/post decrement are defined (overloaded) for **global** and **mstring** variables either together (in binary or the ternary operator) or in connection with other builtin data types. The contents of the **global** array node or **mstring** variable must be compatible with the dominant data type of the operation. If the contents not compatible with the operation (example, incrementing a string of text), the value of the **global** will be interpreted as zero. Examples:

Code Examples	Results

global gbl("gbl");
 	
int i, j=10;
string a = "10", b = "20", c = "30";
char aa[] = "10", bb[] = "20", cc[] = "30";
mstring aaa = "10", bbb = "20", ccc = "30";
 	
gbl.Kill();	

26 If one operand is a numeric type (**long**, **float** etc.), the **mstring** or **global** will be interpreted as a numeric value rather than as a string. If both operands are of type **global** or **mstring**, they will be compared as strings. If one operand is of type **global** or **mstring** and the other is of type **char*** or **string**, they will be compared as strings.

27 The left-hand-side must be of type **mstring** or **global** while the right-hand-side may be of types **mstring**, **global**, **float**, **double**, **int**, **long**, **char***, or **string**. When arithmetic assignment operators are used, right-hand-side **string**, **char***, and **global** operands will be converted to numeric following the default Mumps conversion rules.

28 Note: because the overloaded bitwise *and* operator (&) is of lower precedence than the bit shift operator <<, in output operations (such as when using *cout*), an expression involving the bitwise & operator must to be in parentheses.

gbl(a,b,c) = 10;
gbl(aa,bb,cc) = 20;
gbl(aaa,bbb,ccc) = 30;
 	
i = gbl(a,b,c) + 20;
cout << i << endl;
 	
i = 20 + gbl(a,b,c);
cout << i << endl;
 	
i = gbl(a,b,c) / j;
cout << i << endl;
 	
i = gbl(a,b,c) * 2;
cout << i << endl;
 	
gbl(a,b,c) ++;
cout << gbl(a,b,c) << endl;
 	
gbl(a,b,c) --;
cout << gbl(a,b,c) << endl;
 	
i = ++ gbl(a,b,c);
cout << i << " " << gbl(a,b,c) << endl;
 	
i = gbl(a,b,c) ++;
cout << i << " " << gbl(a,b,c) << endl;
 	
gbl(a,b,c) += 10;
cout << gbl(a,b,c) << endl;
 	
gbl(a,b,c) -= 10;
cout << gbl(a,b,c) << endl;
 	
gbl(a,b,c) *= 2;
cout << gbl(a,b,c) << endl;
 	
gbl(a,b,c) /= 2;
cout << gbl(a,b,c) << endl;
 	
aaa="aaa"; bbb="bbb"; ccc="ccc";
cout << (aaa&bbb&ccc) << endl;
	
	50
	
	50
	
	3
	
	60
	
	31
	
	30
	
	31 31
	
	31 32
	
	42
	
	32
	
	64
	
	32
	
	aaabbbccc

Figure 44 Code Examples	

11.6 Functions for Global and Mstring Objects

As is the case with Mumps functions, characters in strings are counted beginning with one, not zero. Thus, the substring beginning at position 3 through and including position 5 in the string *"abcdef"* is *"cde"*.

If an object of type **mstring** contains a string that is to be used as a global array reference in connection with one of the functions below, the global array reference must be preceded by a circumflex character (^) as is the case in Mumps and, also, the indices must be constants. Example:

```
mstring x="^g(1)";
cout x.Qlength() << endl; // prints 1
```

Function Parameters
INT An expression involving **int**, **long**, **float**, **double**, **mstring** or **global** the result of which can be interpreted as an integer. Data of type **char*** may not be used.

STR An expression involving **int**, **long**, **float**, **double**, **mstring** or **global** the result of which can be interpreted as a string. Data of type **char*** may be used but not as part of an expression.

Function	Description

`int mstring::Ascii( [INT] )`
`int global::Ascii( [INT] )`
	Returns the decimal value of the first ASCII character in the invoking **global** or **mstring**. If an integer argument is given, it returns the decimal value of the character at the offset designated by the argument. **mstring** and **global** arguments will be interpreted as integers.
	
	`mstring s1="abcdef";`
	`s1.Ascii()  -> 97`
	`s1.Ascii(2) -> 98`

`void mstring::Assign(global)`
`void mstring::Assign(mstring)`
`void mstring::Assign(string)`
`void mstring::Assign(char*)`
`void mstring::Assign(int)`
`void mstring::Assign(long)`
`void mstring::Assign(double)`
	Assign a value to the global array reference containg in the invoking **mstring**. Contents of invoking **mstring** must conform to Mumps global array naming conventions and all indices must be constants, global array references, or variables previously defined in the Mumps Interpreter symbol table (see: *SymPut()*). Items placed in the Mumps Interpreter symbol table are discarded when the program ends. This function throws a *MumpsGlobalException* in the event of error.
	
	`mstring x="^g(1,1)";`
	`global g("g");`
	`x.Assign("test test");`
	`cout << g(1,1) << endl; // -> test test`
	
	`SymPut("a","1"); // a put in symTab`
	`x="^g(a,a)";  // reference uses a`
	`x.Assign("abc");`
	`cout << g(1,1) << endl; // -> abc`
	
	`g(1)=1;`
	`x="^g(^g(1),^g(1))";`
	`x.Assign("xyz");`
	`cout << g(1,1) << endl; // -> xyz`

`double global::Avg()`	Returns the average of the values of data bearing nodes beneath the given global array reference.
	
	`global a("a");`
	`for (i=0; i<1000; i++)`
	`  for (j=1; j<10; j++)`
	`  a(i,j) = j;`
	
	`a("100").Avg() -> avg below node a("100")`
	`a().Avg()  -> average of all nodes`

`void global::Centroid(global B)`	A centroid vector B is calculated from the invoking two dimensional **global** array matrix. An element of the centroid vector is the average of the values of each for the corresponding column of the matrix. Any previous contents of the **global** array named to receive the centroid vector are lost. The invoking **global** array must contain at least two dimensions.
	
	`global A("A");`
	`global B("B");`
	`mstring i,j;`
	`for (i=0; i<10; i++)`

	for (j=1; j<10; j++)
	  A(i,j) = 5;
	A().Centroid(B());
	mstring a="";
	while (1) {
	 a=B(a).Order();
	 if (a=="") break;
	 cout << a << " --> " << B(a) << endl;
	 }
	
	Yields:
	
	  1 --> 5
	  2 --> 5
	  3 --> 5
	  4 --> 5
	  5 --> 5
	  6 --> 5
	  7 --> 5
	  8 --> 5
	  9 --> 5

mstring mstring::Concat(char *)
mstring mstring::Concat(global)
mstring mstring::Concat(mstring)
mstring mstring::Concat(string)
mstring mstring::Concat(int)
mstring mstring::Concat(long)
mstring mstring::Concat(double)
 	
mstring global::Concat(string)
mstring global::Concat(global)
mstring global::Concat(char *)
mstring global::Concat(mstring)
mstring global::Concat(int)
mstring global::Concat(long)
mstring mstring::Concat(double)
	Returns **mstring** consisting of the value from the invoking object concatenated with the value of the parameter
	
	mstring a="aaa",b="bbb",c;
	
	c=a.Concat(b); // c contains aaabbb

long global::Count()	Returns the number of data bearing nodes beneath the given global array reference.
	
	global a("a");
	mstring i,j;
	for (i=1; i<11; i++)
	  for (j=1; j<11; j++)
	  a(i,j) = 5;
	a().Count() -> 100
	a("5").Count() -> 10

void global::DocCorrelate(global B,
  mstring fcn, double threshold)
 	
void global::DocCorrelate(global B, char *
  fcn, double threshold)
	DocCorrelate() builds a square *document-document* correlation matrix from the invoking **global** array *document-term matrix*. The name of the function to be used in calculating the *document-document* similarity is given by *fcn* and may be *Cosine*, *Jaccard*, *Dice*, or *Sim1*. The minimum correlation threshold is given in *threshold* which defaults to 0.80 if omitted.
	
	global A("A");
	global B("B");
	
	long i,j;

	A("1","computer")=5; A("1","data")=2; A("1","program")=6; A("1","disk")=3; A("1","laptop")=7; A("1","monitor")=1; A("2","computer")=5; A("2","printer")=2; A("2","program")=6; A("2","memory")=3; A("2","laptop")=7; A("2","language")=1; A("3","computer")=5; A("3","printer")=2; A("3","disk")=6; A("3","memory")=3; A("3","laptop")=7; A("3","USB")=1; A().DocCorrelate(B(),"Cosine",.5); B.TreePrint(); Yields 1 2=0.887096774193548 3=0.741935483870968 2 1=0.887096774193548 3=0.701612903225806 3 1=0.741935483870968 2=0.701612903225806
mstring global::Extract([INT [,INT]]) mstring mstring::Extract([INT [,INT]])	Returns the substring of the invoking **global** or **mstring** beginning at the position designated by the 1st argument and ending at the position designated by the second argument, inclusive. If no second argument is given, the single character designated by the first argument is returned. If the second argument specifies a position beyond the end of the string, the remainder of the string including and following the character designated by the first argument is returned. global g1("g1"); g1("1")="abcdef"; g1("1").Extact(2) -> b g1("1").Extact(2,4) -> bcd g1("1").Extract(2,99) -> bcdef
mstring mstring::Eval()	Evaluates the Mumps expression in the invoking **mstring** object and returns the result in an **mstring**. If an error occurs, an *InterpreterException* is thrown. The invoking **mstring** object may contain a valid mumps expression. mstring x="5*2"; x.Eval() -> 10 global g("g");

	g("1","1")=22; x="^a(1,1)"; x.Eval() -> 22
int global::Find(STR [,INT]) int mstring::Find(STR [,INT])	Searches the invoking string for the firts instance of the **STR** argument and, if **STR** is found, returns the character position of the character immediately following the instance of **STR**. If an **INT** argument is provided, the search begins at that character offset in the invoking string. Returns -1 is **STR** is not found. mstring p="abcdefabcdef"; p.Find("def") -> 7 p.Find("def",5) -> 13
mstring Horolog()	Returns an **mstring** of the form "x,y" where *x* is the number of days since December 31, 1840 and *y* is the number of seconds since midnight.

void global::IDF(double DocCount)	The IDF() function calculates for the invoking **global** array vector the i*nverse document frequency* weight of each term. The vector indices should be words and have as stored values the number of documents in which each word occurs. The document count for each element will be replaced by the calculated IDF value. The IDF is calculated as: l*og2(DocCount/*W_n*)+1* where W_n is the number of documents in which a term appears (the document frequency). The value *DocCount* is the total number of documents present in the collection.
	
	global a("a");
	
	a("now")=2;
	a("is")=5;
	a("the")=6;
	a("time")=3;
	
	a().IDF(4);
	
	a().TreePrint();
	
	Yields:
	
	is=0.678072
	now=2.000000
	the=0.415037
	time=1.415037

mstring global::Justify(INT [,INT]) mstring mstring::Justify(INT [,INT])	Right justifies the invoking object in an **mstring** field whose length is given by the first argument. If the second argument is present and a positive integer, the invoking object is right justified in a field whose length is given by the first argument with the number decimal places as specified by the second argument. The two argument form imposes a numeric interpretation upon the first argument. Rounding occurs in the two argument case. mstring p=123.456 p.Justify(10) -> 123.456 p.Justify(10,2) -> 123.46 p="abcdef"; p.Justify(p,10) -> abcdef
void global::Kill()	Kill (delete) the named **global** array node and all

	descendants. To kill and entire **global** array use:
	
	`global gb("gb");`
	`gb().Kill;`

`int global::Length( [STR ] )`
`int mstring::Length( [STR] )`
	Returns the length of the invoking string. If an argument **STR** is given, the number returned is the number of invoking string segments divided by the argument.
	
	`mstring p="abc & def";`
	`p.Length()    -> 9`
	`p.Length("&") -> 2`

`double global::Max()`	Returns the maximum numeric value of the data bearing nodes beneath the given reference. Non-numeric values are treated as zeros.
	
	`global a("a");`
	`mstring i,j;`
	`for (i=1; i<11; i++)`
	`  for (j=1; j<11; j++)`
	`    a(i,j) = rand()%1000;`
	
	`a().Max()     -> 996 (results will vary)`
	`a("10").Max() -> 932`

`double global::Min()`	Returns the minimum numeric value of the data bearing nodes beneath the given reference. Non-numeric values are treated as zeros.
	
	`global a("a");`
	`mstring i,j;`
	`for (i=1; i<11; i++)`
	`  for (j=1; j<11; j++)`
	`    a(i,j) = rand()%1000;`
	`a().Min()     -> 11 (results will vary)`
	`a("10").Min() -->12`

`void global::Multiply(global, global)`	The invoking global array matrix is multiplied by the first argument global array matrix and the result is placed in the second argument global array matrix. The number of columns of the invoking global array matrix must equal the number of rows of the first argument global array matrix. The resulting matrix (second argument) will have *n* rows and *m* columns where *n* is the number of rows of invoking global array matrix and *m* is the number of columns of the first argument global array matrix.
	
	The contents of the second argument, if any, will be deleted before the operation begins. The data stored at each node in the invoking matrix and the first argument matrix must be numeric. All calculations are performed in double precision arithmetic. Each input matrix must be two dimensional. The output matrix is also two dimensional.
	
	`global d("d");`
	`global e("e");`
	`global f("f");`
	
	`d("1","1")=2; d("1","2")=3;`
	`d("2","1")=1; d("2","2")=-1;`
	`d("3","2")=0; d("3","2")=4;`

	e("1","1")=5; e("1","2")=-2;
	e("1","3")=4; e("1","4")=7;
	e("2","1")=-6; e("2","2")=1;
	e("2","3")=-3; e("2","4")=0;
	
	d().Multiply(e(),f());
	
	f().TreePrint();
	
	Yields:
	1
	 1=-8
	 2=-1
	 3=-1
	 4=14
	2
	 1=11
	 2=-3
	 3=7
	 4=7
	3
	 1=-24
	 2=4
	 3=-12
	 4=0

mstring **global**::Name()	Returns an **mstring** containing of the **global** reference with all variables and expressions in the indices evaluated. global a("a"); mstring b="1",c="2",d="3"; a(b,c,d,c+d).Name() -> a("1","2","3","5")
int global::Pattern(STR) int mstring::Pattern(STR)	Evaluates the invoking string according to the pattern string **STR** (see Mumps documentation) and returns 0 (does not match) or 1 (does match). mstring p=12345; p.Pattern("5N" -> 1
mstring global::Piece(STR, INT [,INT]) mstring mstring::Piece(STR, INT [,INT])	Returns a substring of the invoking object delimited by the instances of the first **STR** argument. The **STR** delimiter divides the invoking object into pieces. The substring returned in the two argument case is the i^{th} substring of the invoking object there *i* is the value of the first **INT** argument. In the three argument form, the string returned begins at the i^{th} piece and ends at the j^{th} piece where *j* is the value of the second **INT** argument. If only one argument is given, *i* is assumed to be 1. mstring p="abc.def.ghi"; p.Piece(".") -> abc p.Piece(".",2) -> def p.Piece(".",2,3) -> def.ghi
int global::Qlength(mstring ref) int mstring::Qlength(char * ref)	Returns the number of subscripts in the global array reference. **mstring** global array references must include the circumflex (^) character.[29] global g("g"); g(1,2,3,4,5).Qlength() -> 5

	`mstring x="^g(1,2,3,4,5,6)";` `x.Qlength() -> 6`

`mstring mstring::Query()`
`mstring global::Query()`
	Returns an object of type **mstring** containing the next global array reference in the data base following the invoking global array reference or the empty string if there are none. The invoking object is either a global array reference or an **mstring** containing a string corresponding to a global array reference. **mstring** global array references must include the circumflex (^) character.[29]
	
	`mstring i,j;`
	`global g("g");`
	`for (i=1; i<10; i++)`
	`  for (j=1; j<10; j++)`
	`    g(i,j)=i+i;`
	
	`g().Query()    -> ^g("1","1")`
	`g(2).Query()   -> ^g("2","1")`
	`g(2,2).Query() -> ^g("2","3")`
	
	`i="^g()"`
	`i.Query() -> ^g("1","1")`
	
	`i=i.Query();`
	`i.Query() -> ^g("1","2")`

`mstring mstring::Qsubscript(int)`
`mstring global::Qsubscript(int)`
	Returns the subscript of a global array reference designated by the argument. **mstring** global array references must include the circumflex (^) character.[29]
	
	`global g("g");`
	`g(9,8,7).Qsubscript(3) -> 7`
	
	`mstring x="^g(9,8,7)";`
	`x.Qsubscript(3) -> 7`

`bool global::ReadLine()`
`bool global::ReadLine(FILE *)`
`bool global::ReadLine(istream &)`
 	
`bool mstring::ReadLine()`
`bool mstring::ReadLine(FILE *)`
`bool mstring::ReadLine(istream &)`
	Reads the next input line into the invoking objsect. If no argument is given *stdin* is used. Otherwise, the inout file is determined by the argument.

`int sw(mstring s, mstring t, [int show_aligns=0, int show_mat=0, int gap=-1, int mismatch=-1, int match=2])`
 	
`int sw(string s, string t, [int show_aligns=0, int show_mat=0, int gap=-1, int mismatch=-1, int match=2])`
 	
`int sw(char *s, char *t, [int show_aligns=0, int show_mat=0, int gap=-1, int mismatch=-1, int match=2])`
	Calculate the *Smith-Waterman* Alignment between strings *s* and *t*. Result returned is the highest alignment score achieved. Parameters other than the first two are optional. If only some of the optional parameters are supplied, only trailing parameters may be omitted, as per C/C++ rules.
	
	If you compare very long strings (>100,000 character), you may exceed stack space. This can be increased under Linux with the command:
	
	`ulimit -s unlimited`
	
	Other options are: `ulimit -a` and `ulimit -aH` to show limits.
	
	If *show_aligns* is zero, no printout of alternative

29 See example in Figure 47 on page 95.

	alignments is produced (default). If *show_aligns* is not zero, a summary of the alternative alignments will be printed. If *show_mat* is zero, intermediate matrices will not be printed (default).
	
	The parameters *gap*, *mismatch* and *match* are the gap and mismatch penalties (normally negative integers) and the match reward (a positive integer). If insufficient memory is available, a *segmentation* violation will be raised.
	
	The first character of each sequence string MUST be blank.
	
	In the printed output, a colon represents a match, a hyphen represents a stretch of the associated string and a blank indicates mismatch.
	
	`char s[]=" now is the time for all good`
	`  men to come to the aid of the party";`
	
	`char t[]=" time  for  good  men";`
	
	`int i=sw(s,t,1,0,-1,-1,3);`
	
	`cout << "Score: " << i << endl;`
	
	`Results in:`
	
	`  12 time- for all good-- men 32`
	`     :::: :::::   :::::  ::::`
	`   1 time  for --  good   men 22`
	
	`score=48`

`int SQL_Command(mstring)`
`int SQL_Command(string)`
`int SQL_Command(char *)`
	Passes the string argument to the SQL database server. See Mumps *sql* command for a description of the argument. The results are written to a file named *mumps.tmp* where columns are <tab> separated.

`int SQL_Connect(char *)`
`int SQL_Connect(string)`
`int SQL_Connect(mstring)`
	Establishes connection with the database server (see Mumps command *sql/d* for a description of the arguments).

`int SQL_Disconnect();`	Disconnects from the database server.

`int SQL_Format()`
`int SQL_Format(mstring)`
`int SQL_Format(string)`
`int SQL_Format(char *)`
	Formats the Mumps **global** array database on the SQL server (see Mumps *sql/d* for a description of the arguments). If no argument is given, system defaults are used.

`mstring SQL_Message()`	Returns most recent SQL database server returned message or the empty string if there is none.
`bool SQL_Msql()`	Returns *true* if the **global** arrays are being stored in a MySQL database server.
`bool SQL_Native()`	Returns *true* if the **global** arrays are being stored in a native database.
`bool SQL_Open()`	Returns *true* if there is a connection to the database server, *false* otherwise.
`bool SQL_Postgres()`	Returns *true* if the **global** arrays are being stored in a PostgreSQL database server.
`mstring SQL_Table()`	Returns an **mstring** containing name of the current

`mstring SQL_Table(mstring, [int] )`
`mstring SQL_Table(string, [int] )`
`mstring SQL_Table(char *, [int] )`
	global array table (default: *mumps*), followed by a comma, followed by the maximum number of columns permitted in the table (default is 10). If arguments are provided, they set the name of the table and the maximum number of columns in the table (maximum of 10). If the second argument is omitted, it defaults to 10.

`double global::Sum()`	The global array nodes beneath the invoking referenced global array are summed. Non-numeric quantities are treated as zero.
	
	`global a("a");`
	`mstring i, j;`
	`for (i = 1; i < 11; i++)`
	`  for (j = 1; j < 11; j++)`
	`   a(i, j) = 5;`
	`cout << a().Sum() << endl;   // -> 500`
	`cout << a("5").Sum() << endl; // -> 50`

`mstring SymGet(T1 name)`	Retrieves the value of the variable whose name is contained in *name* from the Mumps Interpreter symbol table. Throws *MumpsSymbolTableException* if the variable is not found. The data type **T1** may be **global**, **mstring** or **char***. See also: *SymPut().*
	
	`SymPut("k","100");`
	`cout << SymGet("k") << endl; // -> 100`

`bool SymPut(T1 name, T1 value)`	Insert into the Mumps Interpreter symbol table a variable whose name is contained in *name* with the value contained in *value*. The data type **T1** and **T2** may be any combination of **global**, **char*** or **mstring**. Returns *true* if successful, *false* otherwise. Variables in the Mumps Interpreter symbol table may be accessed by expressions passed to the function *mstring::Eval()* or *mstring::Assign()*. See also: *SymGet()*.
	
	`mstring i="3*k";`
	`SymPut("k","100");`
	`cout << i.Eval() << endl; // -> 300`

`void global::TermCorrelate(global B)`	*TermCorrelate()* builds a square *term-term* correlation matrix in **global** array B from the invoking **global** array document-term matrix.
	
	`global A("A");`
	`global B("B");`
	`int main() {`
	`  long i,j;`
	
	`  A("1","computer")=5;`
	`  A("1","data")=2;`
	`  A("1","program")=6;`
	`  A("1","disk")=3;`
	`  A("1","laptop")=7;`
	`  A("1","monitor")=1;`
	
	`  A("2","computer")=5;`
	`  A("2","printer")=2;`
	`  A("2","program")=6;`
	`  A("2","memory")=3;`
	`  A("2","laptop")=7;`
	`  A("2","language")=1;`

```
 A("3","computer")=5;
 A("3","printer")=2;
 A("3","disk")=6;
 A("3","memory")=3;
 A("3","laptop")=7;
 A("3","USB")=1;

 A.TermCorrelate(B);

 mstring a;
 mstring b;

 a="";

 while (1) {
  a=B(a).Order();
  if (a=="") break;
  cout << a << endl;
  b="";
  while (1) {
   b=B(a,b).Order();
   if (b=="") break;
   cout <<"    " << b << "(" << B(a,b)
        << ")" << endl;
   }
  }
 return 0;
 }

Yields:

    USB
    computer(1)
    disk(1)
    laptop(1)
    memory(1)
    printer(1)
computer
    USB(1)
    data(1)
    disk(2)
    language(1)
    laptop(3)
    memory(2)
    monitor(1)
    printer(2)
    program(2)
data
    computer(1)
    disk(1)
    laptop(1)
    monitor(1)
    program(1)
disk
    USB(1)
    computer(2)
    data(1)
    laptop(2)
    memory(1)
    monitor(1)
    printer(1)
    program(1)
```

	language
	    computer(1)
	    laptop(1)
	    memory(1)
	    printer(1)
	    program(1)
	laptop
	    USB(1)
	    computer(3)
	    data(1)
	    disk(2)
	    language(1)
	    memory(2)
	    monitor(1)
	    printer(2)
	    program(2)
	memory
	    USB(1)
	    computer(2)
	    disk(1)
	    language(1)
	    laptop(2)
	    printer(2)
	    program(1)
	monitor
	    computer(1)
	    data(1)
	    disk(1)
	    laptop(1)
	    program(1)
	printer
	    USB(1)
	    computer(2)
	    disk(1)
	    language(1)
	    laptop(2)
	    memory(2)
	    program(1)
	program
	    computer(2)
	    data(1)
	    disk(1)
	    language(1)
	    laptop(2)
	    memory(1)
	    monitor(1)
	    printer(1)

void global::Transpose(global)	The invoking two dimensional matrix **global** object is transposed and the result is placed in two dimensional **global** array object given as the argument. Any prior contents of the output array out are deleted before the operation commences.
	
	global d("d");
	global f("f");
	
	d("1","1")=2;
	d("1","2")=3;
	d("2","1")=4;
	d("2","2")=0;
	d().Transpose(f());

	f.TreePrint(); Results: 1 1=2 2=4 2 1=3 2=0
void global::TreePrint([int, [char]])	Prints the invoking global array as a tree. If a the first **int** argument is given, it is the number of spaces to indent each level (default is 1 if not specified). If the second argument is given, it is the character used to indent (default is blank character). See example in *global::Multiply()* above.
bool ZSeek(FILE *file, mstring offset) bool ZSeek(FILE *file, global offset) bool ZTell(FILE *file)	These functions are used in connection with direct access files opened with **FILE** pointers (see: *fopen()*). They are compatible with 64 bit file systems. *ZSeek()* positions the file designated by *file* to the offset specified in *offset*, a positive integer contained in a variable of type **mstring** or **global**. *ZTell()* places the current file offset in the file designated by *file* to the integer value in the **mstring** or **global** variable represented given by *offset*. Both functions return *true* if successful. Ordinarily, file offsets will be obtained by *ZTell()* and these will be stored in a data base. These values will be subsequently used by *ZSeek()* to reposition the file to the point it was at when the *ZTell()* was performed. After re-positioning, the next input or output operation on the file will occur at the point designated by *offset*. All offsets are positive integers relative to the start of the file.

Figure 45 Functions Defined on **mstring** and **global**

Some Function Examples	Results

char gname[]="doc";
global doc(gname);
 	
doc("1")="abcdef";
mstring ppp = "abcdef";
mstring aaa;
 	
cout << ppp.Ascii() << endl;
cout << doc("1").Ascii() << endl;
cout << ppp.Ascii(1) << endl;
cout << doc("1").Ascii(1) << endl;
 	
cout << ppp.Length() << endl;
cout << doc("1").Length() << endl;
 	
ppp="aaa & bbb";
aaa="&";
 	
cout << ppp.Length("&") << endl;
cout << ppp.Length("*") << endl;
	97
	97
	97
	97
	
	6
	6
	
	2
	1

```
cout << ppp.Length(aaa) << endl;                        2

doc("1")="&";
cout << ppp.Length(doc("1")) << endl;                   2

string strng="&";
cout << ppp.Length(strng) << endl;                      2

ppp = "123abc456abc";
doc("1")="123abc456abc";
doc("9")="abc";
cout << ppp.Find("abc") << endl;                        7
cout << doc("1").Find("abc") << endl;                   7
cout << ppp.Find("abc",5) << endl;                      13
cout << doc("1").Find("abc",5) << endl;                 13
cout << doc("1").Find(doc("9"),5) << endl;              13
strng="abc";
cout << ppp.Find(strng,5) << endl;                      13

cout << Horolog() << endl;                              63815,68346

doc("1").ReadLine();                                    abcdef [input]
cout << "readline global " <<doc("1") << endl;          readline global abcdef

ppp.ReadLine();                                         abcdef [input]
cout << "readline mstring " <<ppp << endl;              readline mstring abcdef

ppp="123";
doc("1")=ppp;
strng="3N";

cout << ppp.Pattern("3N") << endl;                      1

doc("9")="3N";
cout << ppp.Pattern(doc("9")) << endl;                  1
cout << doc("1").Pattern("3N") << endl;                 1

doc("1")="3N";
cout << ppp.Pattern(doc("1")) << endl;                  1

cout << doc("1").Justify(10,2) << endl;                       3.00
cout << doc("1").Justify(10) << endl;                           3N
cout << ppp.Justify(10,2) << endl;                          123.00
cout << ppp.Justify(10) << endl;                               123

cout << doc("1").Data() << endl;                        1

doc("2","3")=123;                                       11
cout << doc("2").Data() << endl;

ppp="abcdef";
mstring off="2";

cout << ppp.Extract(2,3) << endl;                       bc
cout << ppp.Extract(off,off+1) << endl;                 bc
cout << ppp.Extract(2) << endl;                         b
cout << ppp.Extract() << endl;                          a

doc("1")=ppp;
cout << doc("1").Extract(2,3) << endl;                  bc
cout << doc("1").Extract(2) << endl;                    b
cout << doc("1").Extract() << endl;                     a
```

```
ppp=-123.45678;
cout << ppp.Fnumber("P","2") << endl;                (123.46)
cout << ppp.Fnumber("P") << endl;                    (123.457)

doc("1")=-123.45678;
cout << doc("1").Fnumber("P","2") << endl;           (123.46)
cout << doc("1").Fnumber("P") << endl;               (123.45678)

ppp="abc.def.ghi";
cout << ppp.Piece(".",2) << endl;                    def
cout << ppp.Piece(".",2,3) << endl;                  def.ghi

strng=".";
cout << ppp.Piece(strng,2,3) << endl;                def.ghi

doc("9")=strng;
cout << ppp.Piece(doc("9"),2,3) << endl;             def.ghi

doc("1")=".";
cout << ppp.Piece(doc("1"),2) << endl;               def
cout << ppp.Piece(doc("1"),2,3) << endl;             def.ghi

long d=1;
float e=1.0;
int f=1;

doc("9")="abcdef";
cout << doc("9").Ascii(e) << endl;                   97
cout << doc("9").Ascii(f) << endl;                   97
cout << doc("9").Ascii(d+1) << endl;                 98
cout << doc("9").Ascii(e+1) << endl;                 98
cout << doc("9").Ascii(f+1) << endl;                 98

off=1;
cout << doc("9").Ascii(off+d) << endl;               98
cout << doc("9").Ascii(off+e) << endl;               98
cout << doc("9").Ascii(off+f) << endl;               98

mstring g=1;
cout << doc("9").Ascii(off+g) << endl;               98
cout << doc("9").Ascii(off+g) << endl;               98
cout << doc("9").Ascii(off+g) << endl;               98
```

Figure 46 Function Examples

Assume that the following entries have been made into the global array data base:

```
 set ^mesh("A01")="Body Regions"
 set ^mesh("A01","047")="Abdomen"
 set ^mesh("A01","047","025")="Abdominal Cavity"
 set ^mesh("A01","047","025","600")="Peritoneum"
 set ^mesh("A01","047","025","600","225")="Douglas' Pouch"
 set ^mesh("A01","047","025","600","451")="Mesentery"
 set ^mesh("A01","047","025","600","451","535")="Mesocolon"
 set ^mesh("A01","047","025","600","573")="Omentum"
 set ^mesh("A01","047","025","600","678")="Peritoneal Cavity"
 set ^mesh("A01","047","025","750")="Retroperitoneal Space"
 set ^mesh("A01","047","050")="Abdominal Wall"
 set ^mesh("A01","047","365")="Groin"
 set ^mesh("A01","047","412")="Inguinal Canal"
```

```
 set ^mesh("A01","047","849")="Umbilicus"
 set ^mesh("A01","176")="Back"
 set ^mesh("A01","176","519")="Lumbosacral Region"
 set ^mesh("A01","176","780")="Sacrococcygeal Region"
 set ^mesh("A01","236")="Breast"
 set ^mesh("A01","236","500")="Nipples"
 set ^mesh("A01","378")="Extremities"
 set ^mesh("A01","378","100")="Amputation Stumps"
 set ^mesh("A01","378","610")="Lower Extremity"
 set ^mesh("A01","378","610","100")="Buttocks"
 set ^mesh("A01","378","610","250")="Foot"
 set ^mesh("A01","378","610","250","149")="Ankle"
 set ^mesh("A01","378","610","250","300")="Forefoot, Human"
 set ^mesh("A01","378","610","250","300","480")="Metatarsus"
 set ^mesh("A01","378","610","250","300","792")="Toes"
 set ^mesh("A01","378","610","250","300","792","380")="Hallux"
 set ^mesh("A01","378","610","250","510")="Heel"
 set ^mesh("A01","378","610","400")="Hip"
 set ^mesh("A01","378","610","450")="Knee"
 set ^mesh("A01","378","610","500")="Leg"
 set ^mesh("A01","378","610","750")="Thigh"
 set ^mesh("A01","378","800")="Upper Extremity"
 set ^mesh("A01","378","800","075")="Arm"
 set ^mesh("A01","378","800","090")="Axilla"
 set ^mesh("A01","378","800","420")="Elbow"
 set ^mesh("A01","378","800","585")="Forearm"
 set ^mesh("A01","378","800","667")="Hand"
 set ^mesh("A01","378","800","667","430")="Fingers"
 set ^mesh("A01","378","800","667","430","705")="Thumb"
 set ^mesh("A01","378","800","667","715")="Wrist"
 set ^mesh("A01","378","800","750")="Shoulder"
```

```
global mesh("mesh");
mstring x;
int i,j;

x = "^mesh()"; // initial global array reference - beginning of array
x = x.Query(); // find first real reference

while (1) {
 if (x == "") break; // nothing to print
 i = x.Qlength(); // how many subscripts
 for (j=0; j<i; j++) cout << " "; // indent by number of subscripts
 cout << x.Qsubscript(i) << " " << x.Eval() << endl; // show index & value
 x = x.Query(); // get next
 }
```

The above code yields:

```
  047 Abdomen
   025 Abdominal Cavity
    600 Peritoneum
     225 Douglas' Pouch
     451 Mesentery
      535 Mesocolon
     573 Omentum
     678 Peritoneal Cavity
    750 Retroperitoneal Space
   050 Abdominal Wall
   365 Groin
   412 Inguinal Canal
   849 Umbilicus
  176 Back
```

```
  519 Lumbosacral Region
  780 Sacrococcygeal Region
 236 Breast
  500 Nipples
 378 Extremities
  100 Amputation Stumps
  610 Lower Extremity
   100 Buttocks
   250 Foot
    149 Ankle
    300 Forefoot, Human
     480 Metatarsus
     792 Toes
      380 Hallux
    510 Heel
   400 Hip
   450 Knee
   500 Leg
   750 Thigh
  800 Upper Extremity
   075 Arm
   090 Axilla
   420 Elbow
   585 Forearm
   667 Hand
    430 Fingers
     705 Thumb
    715 Wrist
   750 Shoulder
```

Figure 47 Query(), Qsububscript() and Qlength() Example

11.7 Examples

```
#include <fstream>
#include <mumpsc/libmpscpp.h>

global doc("doc");
global idf("idf");
global indx("index");

int main() {

 FILE *u1;

 ofstream u2 ("document-term-matrix-
              weighted.txt", ios::out);
 assert (u2 != 0);

 mstring d,tt,w,null;
 double x,idfmin=6.0;

 null="";

 indx().Kill();

 for (d=doc(null).Order(); d != null; d =
      doc(d).Order() ) {
  u2 << "doc=" << d << "   ";

  for (w = doc(d,null).Order(); w != null;
       w = doc(d,w).Order()) {
```

```
#!/usr/bin/mumps
# weight.mps December 26, 2011

 open 2:"document-term-matrix-
        weighted.txt,new"

 idfmin=6.0;

 kill ^index

 for d=$order(^doc(d)) do

 . use 2 write !,"doc=",d,?15

 . for w=$order(^doc(d,w)) do
```

```
  if (idf(w) < idfmin) {
   doc(d,w).Kill();
   }
  else {
   x = idf(w)*doc(d,w);
   doc(d,w)=x;
   indx(w,d)=x;
   u2 << w << "(" << x << ") ";
   }
  }
 u2 << endl << endl;
 }

u2.close();

ofstream u3
 ("term-document-matrix-weighted.txt",
   ios::out);
assert (u3 != 0);

for (w=indx(null).Order(); w != null;
     w=indx(w).Order()) {
 u3 << w << "   ";
 for (d=indx(w,null).Order(); d != null;
      d=indx(w,d).Order()) {
  u3 << d << "(" << indx(w,d) << ") ";
  }
 u3 << endl << endl;
 }

u3.close();

return 0;

}
```

```
 .. if ^idf<w<idfmin kill ^doc(d,w)


.. else do
 ... set x=^idf(w)*^doc(d,w)
 ... set ^doc(d,w)=x
 ... set ^index(w,d)=x
 ... write w,"(",x,") "
 . write !




close 2

 open 2:"term-document-matrix-
        weighted.txt,new"
 use 2


for w=$order(^index(w)) do

 . write w,?26
 . for d=$order(^index(w,d)) do

 .. write d,"(",^index(w,d),") "

 . write !


 close 2
```

Figure 48 Document Weighting

12 Licenses

12.1 GNU Licenses

12.1.1 GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

<>

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

<>

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those

sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

 3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.
<>
 4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
<>
8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

 10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

      NO WARRANTY

 11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

 12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS
<>
How to Apply These Terms to Your New Programs

 If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

 To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

   <one line to give the program's name and a brief idea of what it does.>
   Copyright (C) 19yy <name of author>

   This program is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2 of the License, or
   (at your option) any later version.

   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

```
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this program; if not, write to the Free Software
   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA


Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

   Gnomovision version 69, Copyright (C) 19yy name of author
   Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
   This is free software, and you are welcome to redistribute it
   under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

 Yoyodyne, Inc., hereby disclaims all copyright interest in the program
 `Gnomovision' (which makes passes at compilers) written by James Hacker.

 <signature of Ty Coon>, 1 April 1989
 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General
Public License instead of this License.
```

12.1.2 GNU Free Documentation License

```
GNU Free Documentation License
Version 1.1, March 2000

 Copyright (C) 2000  Free Software Foundation, Inc.
     59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.


0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other
written document "free" in the sense of freedom: to assure everyone
the effective freedom to copy and redistribute it, with or without
modifying it, either commercially or noncommercially.  Secondarily,
this License preserves for the author and publisher a way to get
credit for their work, while not being considered responsible for
modifications made by others.

This License is a kind of "copyleft", which means that derivative
works of the document must themselves be free in the same sense.  It
complements the GNU General Public License, which is a copyleft
license designed for free software.
```

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
      Copyright (c)  YEAR  YOUR NAME.
      Permission is granted to copy, distribute and/or modify this document
      under the terms of the GNU Free Documentation License, Version 1.1
      or any later version published by the Free Software Foundation;
      with the Invariant Sections being LIST THEIR TITLES, with the
      Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
      A copy of the license is included in the section entitled "GNU
      Free Documentation License".
```

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

12.1.3 GNU LESSER GENERAL PUBLIC LICENSE

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

```
 Copyright (C) 1991, 1999 Free Software Foundation, Inc.
     59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.
```

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

  The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

  This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

  When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

  To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these

rights. These restrictions translate to certain responsibilities for
you if you distribute copies of the library or if you modify it.

  For example, if you distribute copies of the library, whether gratis
or for a fee, you must give the recipients all the rights that we gave
you. You must make sure that they, too, receive or can get the source
code. If you link other code with the library, you must provide
complete object files to the recipients, so that they can relink them
with the library after making changes to the library and recompiling
it. And you must show them these terms so they know their rights.

  We protect your rights with a two-step method: (1) we copyright the
library, and (2) we offer you this license, which gives you legal
permission to copy, distribute and/or modify the library.

  To protect each distributor, we want to make it very clear that
there is no warranty for the free library. Also, if the library is
modified by someone else and passed on, the recipients should know
that what they have is not the original version, so that the original
author's reputation will not be affected by problems that might be
introduced by others.

  Finally, software patents pose a constant threat to the existence of
any free program. We wish to make sure that a company cannot
effectively restrict the users of a free program by obtaining a
restrictive license from a patent holder. Therefore, we insist that
any patent license obtained for a version of the library must be
consistent with the full freedom of use specified in this license.

  Most GNU software, including some libraries, is covered by the
ordinary GNU General Public License. This license, the GNU Lesser
General Public License, applies to certain designated libraries, and
is quite different from the ordinary General Public License. We use
this license for certain libraries in order to permit linking those
libraries into non-free programs.

  When a program is linked with a library, whether statically or using
a shared library, the combination of the two is legally speaking a
combined work, a derivative of the original library. The ordinary
General Public License therefore permits such linking only if the
entire combination fits its criteria of freedom. The Lesser General
Public License permits more lax criteria for linking other code with
the library.

  We call this license the "Lesser" General Public License because it
does Less to protect the user's freedom than the ordinary General
Public License. It also provides other free software developers Less
of an advantage over competing non-free programs. These disadvantages
are the reason we use the ordinary General Public License for many
libraries. However, the Lesser license provides advantages in certain
special circumstances.

  For example, on rare occasions, there may be a special need to
encourage the widest possible use of a certain library, so that it becomes
a de-facto standard. To achieve this, non-free programs must be
allowed to use the library. A more frequent case is that a free
library does the same job as widely used non-free libraries. In this
case, there is little to gain by limiting the free library to free
software only, so we use the Lesser General Public License.

  In other cases, permission to use a particular library in non-free
programs enables a greater number of people to use a large body of

free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

  Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

  The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

  A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

  The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

  "Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

  Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

  1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

  You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

    a) The modified work must itself be a software library.

    b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

    c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

    d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

    (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

  3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

  Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining

where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries,

so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

  13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

  14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

  15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

  If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

  To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the

```
"copyright" line and a pointer to where the full notice is found.


    Copyright (C)

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the library, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the
  library `Frob' (a library for tweaking knobs) written by James Random Hacker.

  , 1 April 1990
  Ty Coon, President of Vice

That's all there is to it!
```

12.2 Perl Compatible Regular Expression Library License

Programs written with the MDH may call upon the Perl Compatible Regular Expression Library. In some cases, this library is distributed with the Mumps Compiler. The PCRE Library is not covered by the GNU GPL/LGPL Licenses but, rather, by the license shownn below. The following is the PCRE license:

```
PCRE LICENCE
------------
PCRE is a library of functions to support regular expressions whose syntax
and semantics are as close as possible to those of the Perl 5 language.
Written by: Philip Hazel
University of Cambridge Computing Service,
Cambridge, England. Phone: +44 1223 334714.
Copyright (c) 1997-2001 University of Cambridge
Permission is granted to anyone to use this software for any purpose on any
computer system, and to redistribute it freely, subject to the following
restrictions:
1. This software is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
2. The origin of this software must not be misrepresented, either by
   explicit claim or by omission. In practice, this means that if you use
   PCRE in software which you distribute to others, commercially or
   otherwise, you must put a sentence like this
     Regular expression support is provided by the PCRE library package,
     which is open source software, written by Philip Hazel, and copyright
     by the University of Cambridge, England.
   somewhere reasonably visible in your documentation and in any relevant
   files or online help data or similar. A reference to the ftp site for
```

```
   the source, that is, to
     ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/
   should also be given in the documentation.
3. Altered versions must be plainly marked as such, and must not be
   misrepresented as being the original software.
4. If PCRE is embedded in any software that is released under the GNU
   General Purpose Licence (GPL), or Lesser General Purpose Licence (LGPL),
   then the terms of that licence shall supersede any condition above with
   which it is incompatible.
The documentation for PCRE, supplied in the "doc" directory, is distributed
under the same terms as the software itself.
End
```

Alphabetical Index